Introduction to the M-file Connexions Modules

MATLAB has emerged as a widely used computational tool in many fields of engineering. MATLAB consists of a programming language used in an interactive computing environment that supports the development of programs to solve complex problems. The MATLAB language has become a defacto standard that is also used by several other computational packages, including LabVIEW MathScript and Octave. Generically, we refer to these packages as **m-file environments** because the program files typically are identified by an extension of "m".

The Connexions modules in this course are intended to introduce freshman engineering students to problem solving using an m-file environment. Most of the information in these modules applies to any m-file environment (MATLAB, LabVIEW MathScript, Octave, etc.). There are some differences between environments, and occasionally some material will be specific to a given environment. This material is offset from the surrounding text and labeled with the appropriate environment. For example:

Note: Matlab is a commercial product of The MathWorks.

Note: LabVIEW MathScript is a commercial product of National Instruments.

Note: Octave is an open source environment that is available without charge. Information about Octave is available at the Octave home page.

Finding Help for M-file Environments

There is a wealth of information available about using most m-file environments. In fact, the amount of information may overwhelm someone who is beginning.

The `help` command can be used to get information about a specific command or function. For example, typing `help cos` will give information about the cosine function. Typing `help` will give general information.

A significant amount of information is available on the World Wide Web:

Note: Useful information is available at the MATLAB Helpdesk page, including reference material for MATLAB's functions.

Note: Useful information is available at the National Instruments LabVIEW MathScript Portal, including an interactive demonstration of MathScript.

Note: The definitive source for information is The official GNU Octave Manual. Also, here is one of many good tutorials.

Problem Solving Using M-file Environments

The purpose of this module is to introduce the engineering problem solving process in the context of using m-file environments to solve problems. Many variations of this process exist and no single variation is best for solving all problems. In this module we describe a variation of the engineering problem solving process that applies to m-file environments problem solving. Other variations are described in the reference at the end of this module.

The following problem solving process is fairly involved and may be an excessive amount of work for simple problems. For problems where the solution is straight forward, simply solve the problem; for more complex problems, the solution will usually not be obvious and this process will aid in development of an appropriate solution.

This specific process is divided into a set of seven steps. Each step includes questions that help move you successfully through the problem solving process.

1. Define the Problem

    - What problem are you trying to solve?
    - "What would success look like?"
    - What should the program output? Computed values? A plot or series of plots?

2. Identify given information.

    - What constants or data are supplied?
    - What theory, principles, models and equations have you been given?

3. Identify other available information.

    - What theory, principles, models and equations can you find in other sources (text books, lecture notes, etc.)?

4. Identify further needed information.

    - What other information do you need?
    - Where will you find it?

5. Design and implement your solution to the problem.

    - How can you break the larger problem into smaller problems?
    - Look at the problem from the top down or bottom up?
    - What programming techniques might you use to convert input to output?
    - What variables do you need? Vectors? Arrays?
    - What principles and equations apply to convert input to output?

6. Verify your solution.

    - How do you know your solution is correct?

7. Reflect on your solution.

    - What worked?
    - What didn't?

When solving simple problems you may be able to follow these steps in order. For more complex problems, you may be working on step 5 and realize you need more information. You might then go back to steps 3 or 4 to re-evaluate and find missing information.

Reference: H. Scott Fogler, Steven E. LeBlanc. Strategies for Creative Problem Solving, Prentice Hall, 1995.

Exercise for Problem Solving with M-Files

Exercise:

Problem:

You are part of a design team that is developing a commercial aluminum can crusher. Your preliminary crusher design includes a collection chamber in which cans are collected until a desired weight of cans has accumulated; the cans are then crushed by a hydraulic ram. In preliminary research, you determine that the typical aluminum can is a cylinder with diameter 2.5" and height 4.8" and weighs approximately 15 grams. You have been assigned to model the relationship between the size of the chamber and the weight of (uncrushed) cans it would hold.

Visual Tools for Problem Solving with M-Files

Using an abstract visual representation while developing a program structure is often a useful technique. Several different visual representations have been developed; one of the most comprehensive is UML. Two of the simplest are introduced in this module: flow charts and pseudo code. In both flow charts and pseudo code, elements of the problem solution are described using natural language statements that are visually arranged to show the structure of the program.

A flow chart represents elements of the solution to a problem as statements enclosed in boxes; the sequence in which use elements are performed is identified by arrows connecting the boxes. [link] shows an example flow chart.

Pseudo code represents the elements of the solution to a problem as a sequence of statements. The statements are formatted to show the logical structure of the solution. [link] shows an example of pseudo code.

The following example demonstrates the use of flow charts and pseudo code to develop the structure of a program that solves an engineering problem.

Example:

The ACME Manufacturing Company is planning to manufacture widgets. There are two different manufacturing processes: one cost $10,000 to implement and can manufacture up to 1000 widgets, while the other cost $100,000 to implement and can manufacture up to one million widgets. In addition to the manufacturing cost, there is a fixed cost of $1 per widget (for packing and shipping each widget to the customer). Consider the problem of calculating the cost per unit to manufacture and ship a given number of widgets.

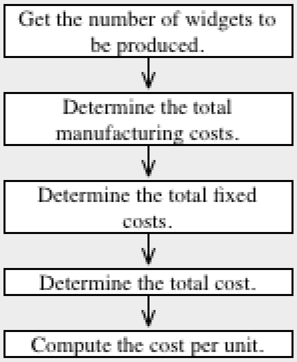
One way to solve the problem is to complete the following steps:

- Get the number of widgets to be produced.
- Determine the total manufacturing costs.

- Determine the total fixed costs.
- Determine the total cost.
- Compute the cost per unit.

[link] shows a flow chart that represents these steps. [link] shows the pseudo code that represents these steps.

First Flow Chart


Get the number of widgets to be produced.
Determine the total manufacturing costs.
Determine the total fixed costs.
Determine the total cost.
Compute the cost per unit.

A flow chart for the widget production problem.

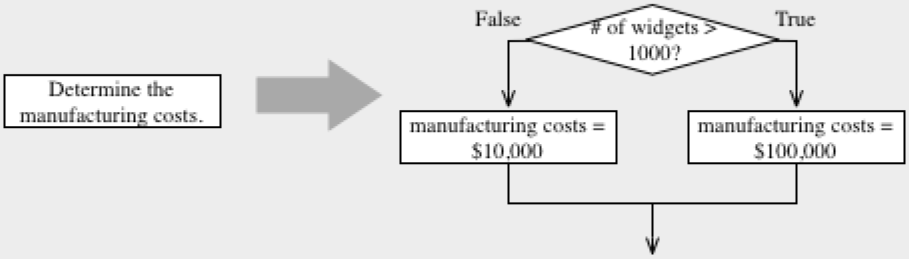
First Pseudo Code

```
Get the number of widgets to be produced.
Determine the manufacturing costs.
Determine the total fixed costs.
Determine the total cost.
Compute the cost per unit.
```

Pseudo code for the widget production problem.

Having developed an initial solution, we can refine those elements whose implementation may not yet be fully defined. In this example, the manufacturing cost depend on the number of widgets to be made; if this number is less than or equal to 1000, the cost is $10,000, while if the number is greater than 1000, the cost is $100,000. We can represent this using the flow chart blocks in [link]. The diamond is a conditional; the branch of the flow chart that is actually executed depends on whether the conditional is true or false.

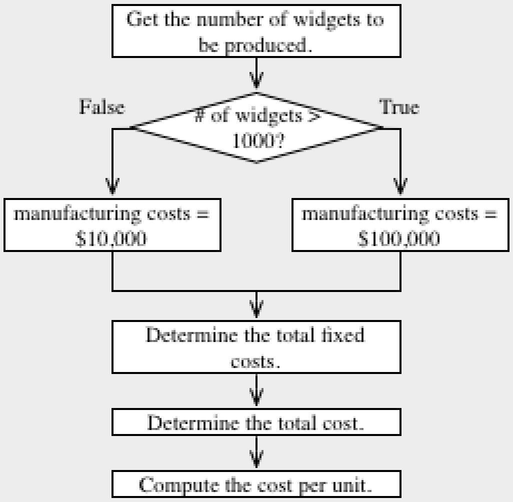
Flow Chart for Manufacturing Costs


Determine the manufacturing costs.
False
# of widgets > 1000?
True
manufacturing costs = $10,000
manufacturing costs = $100,000

A flow chart for the manufacturing costs.

Putting this conditional into the complete flow chart gives the flow chart in [link]. The conditional is incorporated in the the pseudo code to give the pseudo code in [link].

Complete Flow Chart


Get the number of widgets to be produced.
False
# of widgets > 1000?
True
manufacturing costs = $10,000
manufacturing costs = $100,000
Determine the total fixed costs.
Determine the total cost.
Compute the cost per unit.

The complete flow chart for the problem.

Complete Pseudo Code

```
Get the number of widgets to be produced.
if number of widgets > 1000
```

```
    manufacturing costs are $100,000
else
    manufacturing costs are $10,000
Determine the total fixed costs.
Determine the total cost.
Compute the cost per unit.
```

Pseudo code for the widget production problem.

There are advantages and disadvantages for both flow charts and pseudo code. Advantages of using a flow chart include that it provides a strong visual representation of the program and that it is straightforward for novice programmers to use. The primary disadvantage of using a flow chart is that it is possible to create a flow chart that can only be implemented by "spaghetti code". Spaghetti code is considered extremely bad form for many reasons; in particular, it is hard to understand, debug and modify. The primary advantage of pseudo code is that its structure is clearly related to the available control structures in modern computer languages. Pseudo code has several disadvantages: it is not a very strong visual representation, and it is less straightforward for novice programmers.

Basic Mathematical Operations

Operations and Expressions

An m-file environment has all of the standard arithmetic operations (addition, subtraction, etc.) and functions (sine, cosine, logarithm, etc.). The table lists the most commonly used operations; in this table, `x` and `y` are **scalars**. (A scalar is a single value, as opposed to a vector or matrix which consists of many values.)

Operation	m-file
$x - y$	`x-y`
$x + y$	`x+y`
xy	`x*y`
$\frac{x}{y}$	`x/y`
x^y	`x^y`
e^x	`exp(x)`
$\mathrm{log10}\left(x\right)$	`log10(x)`
$\ln\left(x\right)$	`log(x)`
$\mathrm{log2}\left(x\right)$	`log2(x)`
$\cos(x)$	`cos(x)`

Operation	**m-file**
$\sin(x)$	`sin(x)`
$\sqrt{x}$	`sqrt(x)`

Some Common Scalar Mathematical Operations

Expressions are formed from numbers, variables, and these operations. The operations have different precedences. The `^` operation has the highest precedence; `^` operations are evaluated before any other operations. Multiplication and division have the next highest precedence, and addition and subtraction have the lowest precedence. Precedence is altered by parentheses; expressions within parentheses are evaluated before expressions outside parentheses.

Example:
The Table below shows several mathematical formulas, the corresponding expressions, and the values that are computed for the expressions.

formula	**MATLAB Expression**	**Computed Value**
5^2+4^2	`5^2+4^2`	`41`
$(5+4)^2$	`(5+4)^2`	`81`
$\frac{2+3}{4-5}$	`(2 + 3)/(4 - 5)`	`-5`
$\mathrm{log10}\,(100)$	`log10(100)`	`2`

formula	MATLAB Expression	Computed Value
$\ln(4 \times (2+3))$	`log(4*(2+3))`	`2.9957`

Example Expressions

Useful Tricks

These tricks are occasionally useful, especially when you begin programming with m-files.

- A **semicolon** added at the end of a line suppresses the output.
- Often it is useful to **split input** over multiple lines. To split a statement across multiple lines, enter three periods `...` at the end of the line to indicate it continues on the next line.

Example:
Splitting the expression $\frac{2+3}{4-5}$ over multiple lines. `(2+3)... /(4-5)`

Variables in M-file Environments

Variables

A variable is a named storage location that can be set to a particular value which can be used in subsequent computations. For example, we store a value of 5 in the variable `a` with the statement `a=5`. This value remains in `a` until we store a different value (for example, using the command `a=100`) or we clear `a` using the command `clear a`. Once a variable is set to a particular value, we can get this value by using the variable name in an expression (e.g. `a/2`).

Example:
Suppose we wish to compute the circumference of a circle of diameter 5 units using the formula $c = \pi d$. We could first set the variable `d` to a value of 5: `>> d = 5 d = 5.000` Then we could compute the circumference and assign its value to the variable `c`: `>> c = pi*d c = 15.708` In this command, the product of the value of `d` (which is known because we earlier set it to 5) and the value of `pi` (which is a pre defined variable) is computed and the value of the product is stored in the variable `c`.

Variable names must begin with an upper- or lower-case letter. They may contain letters, digits, and underscores; they may not contain spaces or punctuation characters. Variable names are case sensitive, so `A` and `a` are different variables.
Exercise:

Problem:
Valid variable names

Which of the following are valid variable names?

1. `a`

2. `B`
3. `ecky_ecky_ecky_ecky_ptang_zoo_boing`
4. `ecky ecky ecky ecky ptang zoo boing`
5. `2nd`
6. `John-Bigboote`

Solution:

1. Valid.
2. Valid.
3. Valid.
4. Invalid, because the variable name contains spaces.
5. Invalid, because the variable name begins with a number.
6. Invalid, because the variable name contains a dash.

There are several predefined variables. The most commonly used include

- `ans` - the default variable in which computation results are stored.
- `pi` - π.
- `i` or `j` - $\sqrt{-1}$.

Once assigned, variable names remain until they are reassigned or eliminated by the `clear` command.

Variables can contain several types of numerical values. These types include the following:

- Scalar - a scalar is a single value (i.e. a number). `c` and `d` in [link] are scalar variables.
- Vector - a vector is an ordered series of numbers.
- Matrices - a matrix is a rectangular array of numbers. The ability to do computations on vectors and matrices gives MATLAB its name (MATrix LABoratory).
- strings - variables may also contain strings of characters.

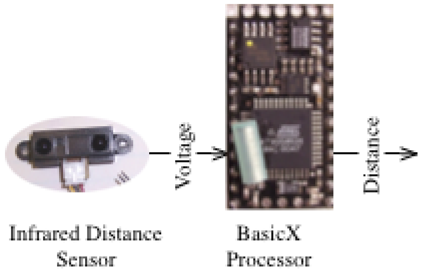
Exercises for Basic Mathematical Operations

Exercise:

Problem:

[link] shows a Sharp GP2D12 infrared distance sensor and a BasicX-24 microprocessor.

Distance Sensor and Processor



The infrared distance sensor and microprocessor.

The distance sensor uses a beam of infrared light to measure the distance from the sensor to an object; the sensor provides an output voltage that has a fairly complicated relationship to this distance. The BasicX processor converts the voltage from the sensor into a number between zero and one. Let us denote this number as x, and the distance (measured in inches) between the sensor and object as d. The relationship between x and d is

Equation:

$$d = \frac{\frac{34.63}{x} - 5.162}{2.54}$$

Compute the value of d for the following values of x:

- $x = 0.10$
- $x = 0.15$

- $x = 0.20$

Exercise:

Problem:

The terminal velocity reached by a sky diver depends on many factors, including their weight, their body position as they fall, and the density of the air through which they fall. The terminal velocity is given by

Equation:

$$V_t = \sqrt{\frac{2mg}{rAC_d}}$$

where

- m is the sky diver's mass
- g is Earth's gravitational constant
- r is the atmospheric density
- A is the sky diver's effective area
- C_d is the sky diver's coefficient of drag

Compute the terminal velocity of the sky diver for each of the following values of m:

- $m = 40$ kg
- $m = 80$ kg
- $m = 120$ kg

Use the following values for the other variables:

- $g = 9.8$
- $r = 1.2$
- $A = 0.5$
- $C_d = 1$

Vectors and Arrays in M-File Environments

Vectors and Arrays in M-File Environments

One significant capability of environments accounts for much of their popularity among engineers: their ability to do vector and matrix computations. M-file environments can operate on the following types of values:

- **Scalar** a scalar is a single value (i.e. a number).
- **Vector** a vector is an ordered series of numbers.
- **Matrix** a matrix is a rectangular array of numbers.

> **Note:**The ability to do computations on vectors and matrices gives MATLAB its name (MATrix LABoratory).

- **String** variables may also contain strings of characters.

Vector Basics

There are several ways to create a vector of values. One is to enclose the values in square brackets. For example, the command `[9 7 5 3 1]` creates the vector of values 9, 7, 5, 3, and 1. This vector can be assigned to a variable `v`: `>> v = [9 7 5 3 1] v = 9 7 5 3 1`

A second way to create a vector of values is with the sequence notation `start:end` or `start:inc:end`. For example, `1:10` creates the vector of integers from 1 to 10: `>> 1:10 ans = 1 2 3 4 5 6 7 8 9 10` The command `1:0.1:2` creates the vector `>> 1:0.1:2 ans = 1.0000 1.1000 1.2000 1.3000 1.4000 1.5000 1.6000 1.7000 1.8000 1.9000 2.0000` The command `10:-1:1` creates the vector `>> 10:-1:1 ans = 10 9 8 7 6 5 4 3 2 1`

Vector elements are accessed using numbers in parentheses. For example if the vector `v` is defined as `v = [9 7 5 3 1]`, the second element of `v` can be accessed as `>> v(2) ans = 7` The fourth element of `v` can be changed as follows: `>> v(4) = 100 v = 9 7 5 100 1`

Element by Element Operations on Vectors

In addition to vector and matrix arithmetic, many operations can be performed on each element of the vector. The following examples use the vector `v = [9 7 5 3 1]`.

- **Addition** the command `v+val` adds `val` to each element of `v`: `>> v+5 ans = 14 12 10 8 6`
- **Subtraction** the command `v-val` subtracts `val` from each element of `v`: `>> v-5 ans = 4 2 0 -2 -4`
- **Multiplication** the command `v*val` multiplies each element of `v` by `val`: `>> v*5 ans = 45 35 25 15 5`
- **Division** the command `v/val` divides each element of `v` by `val`: `>> v/5 ans = 1.80000 1.40000 1.00000 0.60000 0.20000` The command `val./v` divides `val` by each element of `v`: `>> 5./v ans = 0.55556 0.71429 1.00000 1.66667 5.00000`
- **Exponentiation** the command `v.^val` raises each element of `v` to the `val` power: `>> v.^2 ans = 81 49 25 9 1`

More Information on Vectors and Matrices

An excellent tutorial on how to use MATLAB's vector and array capabilities is at the Mathworks MATLAB tutorial page.

One useful method of accessing entire rows or entire columns of the matrix is not mentioned in the tutorial. Suppose that the matrix `A` is defined as `>> A = [1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20] A = 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20` An entire row of `A` can be obtained by specifying a single ":" as the column index: `>> A(2,:) ans = 6 7 8 9 10` Similarly, an entire column of `A` can be obtained by specifying a single ":" as the row index: `>> A(:,3) ans = 3 8 13 18`

Basic Complex and Matrix Operations

Complex numbers

m-file environments have excellent support for complex numbers. The imaginary unit is denoted by `i` or (as preferred in Electrical Engineering) `j`. To create complex variables $z_1 = 7 + i$ and $z_2 = 2e^{i\pi}$ simply enter `z1 = 7 + j` and `z2 = 2*exp(j*pi)`

The table gives an overview of the basic functions for manipulating complex numbers, where z is a complex number.

	m-file
Re(z)	`real(z)`
Im(z)	`imag(z)`
$\lvert z \rvert$	`abs(z)`
Angle(z)	`angle(z)`
z^*	`conj(z)`

Manipulating complex numbers

Operations on Matrices

In addition to scalars, m-file environments can operate on matrices. Some common matrix operations are shown in the Table below; in this table, `M` and `N` are matrices.

Operation	**m-file**
MN	`M*N`
M^{-1}	`inv(M)`
M^T	`M'`
$\det(M)$	`det(M)`

Common matrix operations

Some useful facts:

- The functions `length` and `size` are used to find the dimensions of vectors and matrices, respectively.
- Operations can also be performed on each element of a vector or matrix by proceeding the operator by ".", e.g `.*`, `.^` and `./`.

Example:

Let $A = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$. Then `A^2` will return $\mathrm{AA} = \begin{pmatrix} 2 & 2 \\ 2 & 2 \end{pmatrix}$, while `A.^2` will return $\begin{pmatrix} 1^2 & 1^2 \\ 1^2 & 1^2 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$.

Example:
Given a vector x, compute a vector y having elements $y(n) = \frac{1}{\sin(x(n))}$.
This can be easily be done the command `y=1./sin(x)` Note that using `/` in place of `./` would result in the (common) error "`Matrix dimensions must agree`".

Introduction to Graphing in M-File Environments

One of the reasons that m-file environments are extensively used by engineers is their capability to provide graphical representations of data and computed values. In this module, we introduced the basics of graphing data in m-file environments through a series of examples. This module uses some fundamental operations on vectors that are explained in Vectors and Arrays in M-File Environments.

Example:
The table below shows speed as a function of distance for a braking Dodge Viper decelerating from 70MPH to 0MPH.

Note: This data was not measured; it was computed using the stopping distance reported for a Dodge Viper and assuming constant deceleration. Thus, it may not accurately reflect the braking characteristics of a real Dodge Viper.

Distance (ft)	Velocity (ft/s)
0	102.7
29.1	92.4
55.1	82.1
78.0	71.9
97.9	61.6

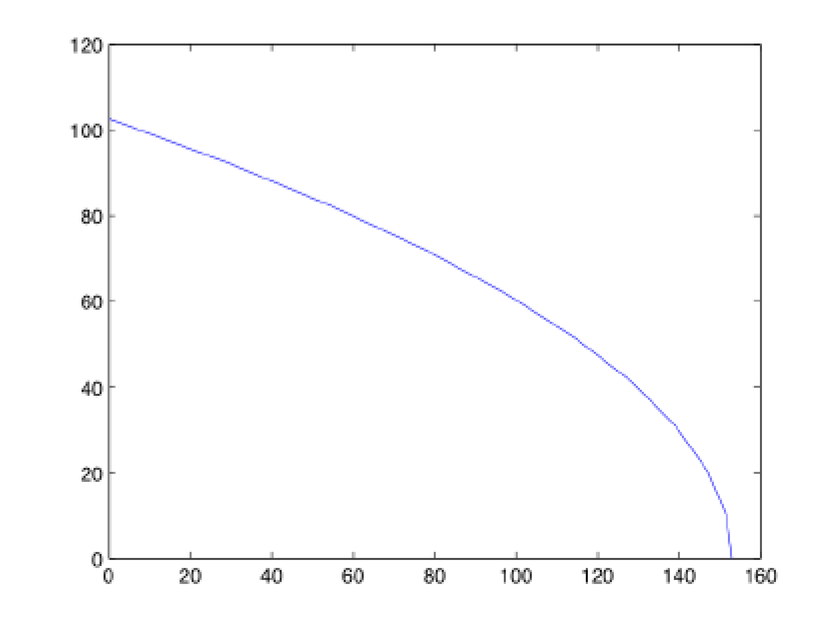
114.7	51.3
128.5	41.1
139.2	30.8
146.9	20.5
151.5	10.3
153.0	0.0

Dodge Viper Stopping Data

The following commands will create a graph of velocity as a function of distance:

```
dist = [0 29.1 55.1 78.0 97.9 114.7 128.5 139.2 146.9 151.5 153.0] vel = [102.7 92.4 82.1 71.9 61.6 51.3 41.1 30.8 20.5 10.3 0.0]
plot(dist,vel)
```

[link] shows the graph created by these commands.

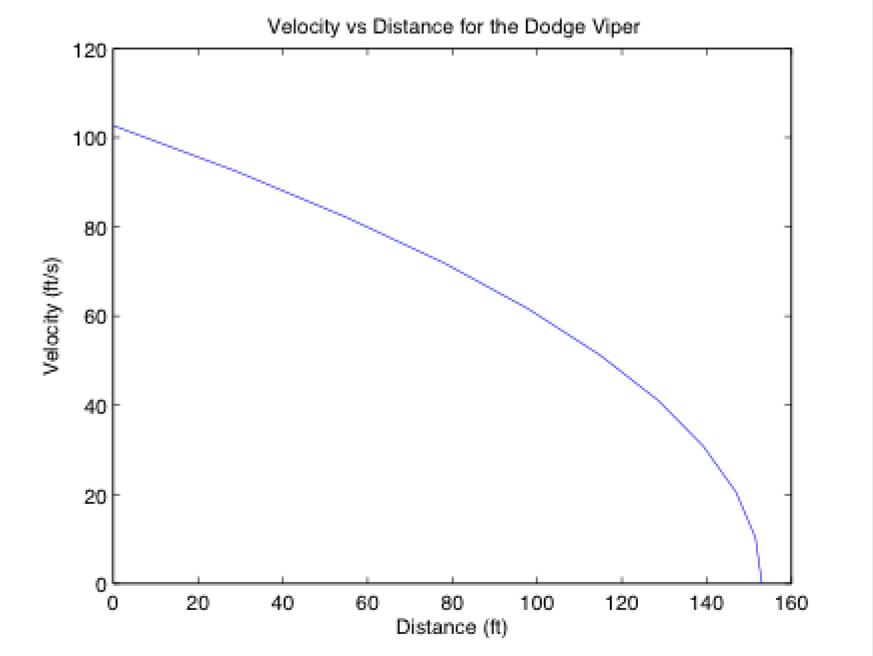


Graph of the Viper's velocity as a function of distance.

This graph shows the data, but violates several important conventions of engineering practice. The axes are not labeled with quantity and units, and the graph does not have a title. The following commands, when executed after the plot command, will label the axes and place a title on the graph.

```
xlabel('Distance (ft)') ylabel('Velocity (ft/s)')
title('Velocity vs Distance for the Dodge Viper')
```

The results of these commands are shown in [link].



Graph of the Viper's velocity as a function of distance. The graph has a title and labels on the axes.

After creating a figure, you may wish to insert it into a document. The method to do this depends on the m-file environment, the document editor and the operating system you are using.

Note: To paste a figure into a Word document on Windows, pull down the **Edit** menu of the window of the MATLAB figure you wish to paste, and select **Copy Figure**. Then go to the Word document into which you wish to insert the figure and use the paste command.

Exercise:

Problem: Repeat [link] using the following data for a Hummer H2:

Note: As in [link], this data was not measured; it was computed using the stopping distance reported for a Hummer H2 and assuming constant deceleration.

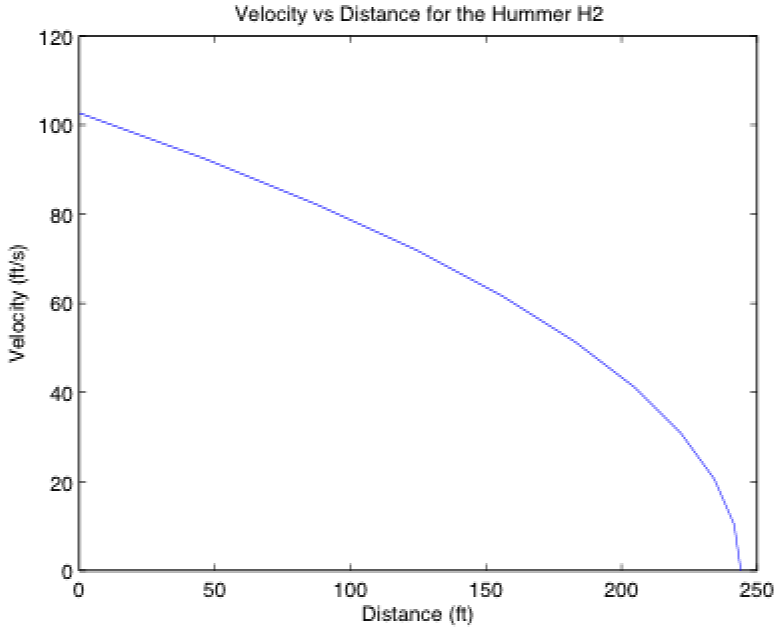
Distance (ft)	Velocity (ft/s)
0	102.7
46.3	92.4
87.8	82.1
124.4	71.9
156.1	61.6
182.9	51.3
204.9	41.1

222.0	30.8
234.2	20.5
241.5	10.3
244.0	0.0

Hummer H2 Stopping Data

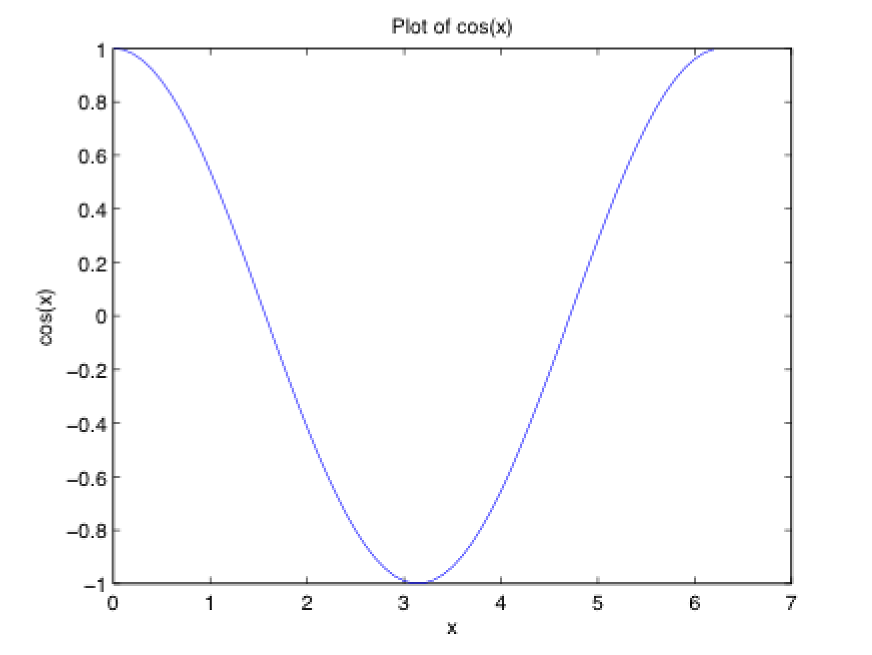
Solution:

[link] shows the graph of the Hummer H2 stopping data.



Graph of the H2 velocity as a function of distance.

Example:
An m-file environment can also be used to plot functions. For example, the following commands plot $\cos(x)$ over one period. `x = 0:0.1:2*pi; y=cos(x) plot(x,y) xlabel('x') ylabel('cos(x)') title('Plot of cos(x)')` [link] shows the graph created by these commands.



Graph of one period of the cosine function.

Exercise:

Problem:

The module Exercises for Basic Mathematical Operations describes how to compute the terminal velocity of a falling sky diver. Plot the terminal velocity as a function of the sky diver's weight; use weights from 40kg to 500kg.

Exercise:

Problem:

In electrical circuit analysis, the equivalent resistance R_{eq} of the parallel combination of two resistors R_1 and R_2 is given by the equation

Equation:

$$R_{\text{eq}} = \frac{1}{\frac{1}{R_1} + \frac{1}{R_2}}$$

Set $R_2 = 1000\ \text{Ohms}$ and plot R_{eq} for values of R_1 from 100 Ohms to 3000 Ohms.

Exercise:

Problem:

In an experiment, a small steel ball is dropped and videoed against a checkered background. The video sequence is analyzed to determine the height of the ball as a function of time to give the data in the following table:

Time (s)	Height (in)
0.0300	22.0
0.0633	21.5
0.0967	20.5
0.1300	18.8

0.1633	17.0
0.1967	14.5
0.2300	12.0
0.2633	8.0
0.2967	3.0

Height and Time Data

This experimental data is to be compared to the theoretically expected values given by the following equation:
Equation:

$$h = 22\text{ in} - \frac{1}{2}gt^2$$

where h is in inches, t is in seconds, and $g = 386.4\frac{\text{in}}{s^2}$. Create a graph that compares the measured data with the theoretically expected values; your graph should conform to good conventions for engineering graphics. Plot the measured data using **red circles**, and plot the theoretically expected values using a **blue line**.

Graphical representation of data in MATLAB

Graphical representation of data in MATLAB

MATLAB provides a great variety of functions and techniques for graphical display of data. The flexibility and ease of use of MATLAB's plotting tools is one of its key strengths. In MATLAB graphs are shown in a figure window. Several figure windows can be displayed simultaneously, but only one is active. All graphing commands are applied to the active figure. The command `figure(n)`will activate figure number `n` or create a new figure indexed by `n`.

Tools for plotting

In this section we present some of the most commonly used functions for plotting in MATLAB.

- `plot`- The plot and stem functions can take a large number of arguments, see help plot and help stem. For example the line type and color can easily be changed. `plot(y)` plots the values in vector `y`versus their index. `plot(x,y)` plots the values in vector `y`versus `x`. The `plot` function produces a piecewise linear graph between its data values. With enough data points it looks continuous.
- `stem`- Using `stem(y)`the data sequence `y`is plotted as stems from the x-axis terminated with circles for the data values. `stem` is the natural way of plotting sequences. `stem(x,y)` plots the data sequence `y` at the values specified in `x`.
- `xlabel('string')`- Labels the x-axis with `string`.
- `ylabel('string')`- Labels the y-axis with `string`.
- `title('string')`- Gives the plot the title `string`.

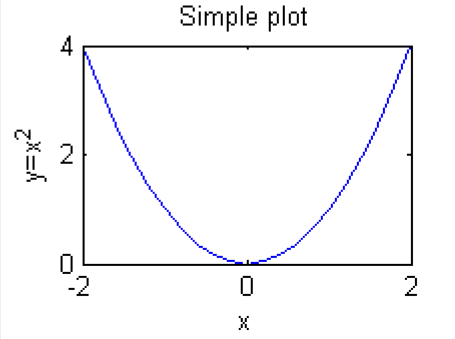
To illustrate this consider the following example.

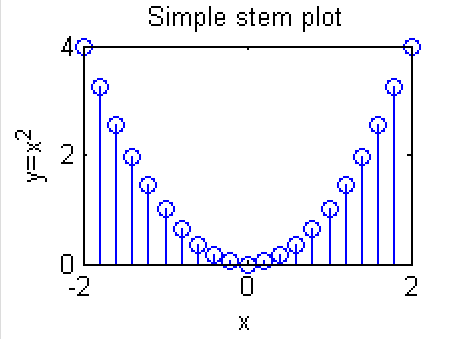
Example:

In this example we plot the function y = x2 for x 2 [-2; 2].

```
x = -2:0.2:2;
y = x.^2;
figure(1);
plot(x,y);
xlabel('x');
ylabel('y=x^2');
title('Simple plot');
figure(2);
stem(x,y);
xlabel('x');
ylabel('y=x^2');
title('Simple stem plot');
```

This code produces the following two figures.





Some more commands that can be helpful when working with plots:

- hold on / off - Normally hold is off. This means that the plot command replaces the current plot with the new one. To add a new plot to an existing graph use `hold on`. If you want to overwrite the current plot again, use `hold off`.

- `legend('plot1','plot2',...,'plot N')`- The `legend` command provides an easy way to identify individual plots when there are more than one per figure. A legend box will be added with strings matched to the plots.
- `axis([xmin xmax ymin ymax])`- Use the `axis` command to set the axis as you wish. Use `axis on/off` to toggle the axis on and off respectively.
- `subplot(m,n,p)` -Divides the figure window into `m` rows, `n` columns and selects the `p`p'th subplot as the current plot, e.g `subplot(2,1,1)` divides the figure in two and selects the upper part. `subplot(2,1,2)` selects the lower part.
- `grid on/off` - This command adds or removes a rectangular grid to your plot.

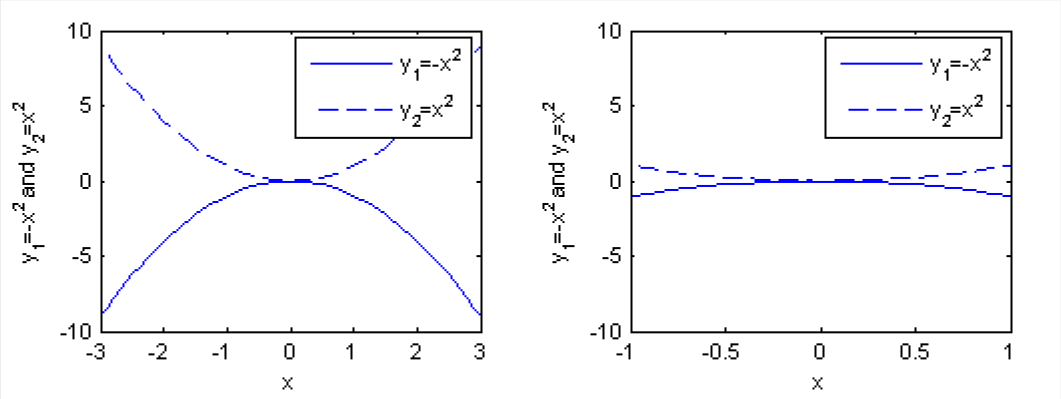
Example:

This example illustrates `hold`, `legend` and `axis`.

```
x = -3:0.1:3; y1 = -x.^2; y2 = x.^2;
figure(1);
plot(x,y1);
hold on;
plot(x,y2,'--');
hold off;
xlabel('x');
ylabel('y_1=-x^2 and y_2=x^2');
legend('y_1=-x^2','y_2=x^2');
figure(2);
plot(x,y1);
hold on;
plot(x,y2,'--');
hold off;
xlabel('x');
ylabel('y_1=-x^2 and y_2=x^2');
```

```
legend('y_1=-x^2','y_2=x^2');
axis([-1 1 -10 10]);
```

The result is shown below.

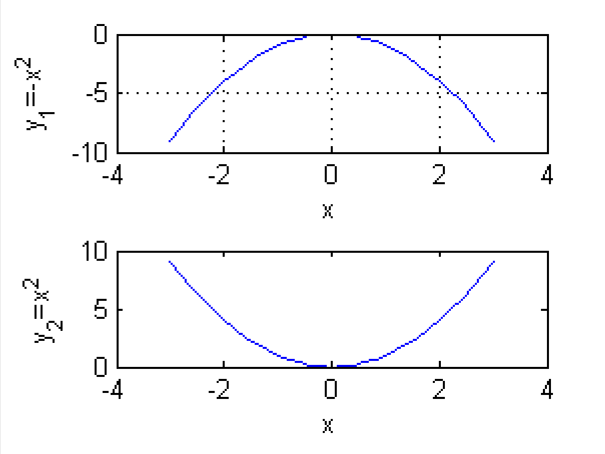


Example:

In this example we illustrate subplot and grid.

```
x = -3:0.2:3; y1 = -x.^2; y2 = x.^2;
subplot(2,1,1);
plot(x,y1);
xlabel('x'); ylabel('y_1=-x^2');
grid on;
subplot(2,1,2);
plot(x,y2);
xlabel('x');
ylabel('y_2=x^2');
```

Now, the result is shown below.



Printing and exporting graphics

After you have created your figures you may want to print them or export them to graphic files. In the "File" menu use "Print" to print the figure or "Save As" to save your figure to one of the many available graphics formats. Using these options should be sufficient in most cases, but there are also a large number of adjustments available by using "Export setup", "Page Setup" and "Print Setup".

To streamline the graphics exportation, take a look at exportfig package at Mathworks.com, URL: http://www.mathworks.com/matlabcentral/fileexchange/loadFile.do?objectId=727.

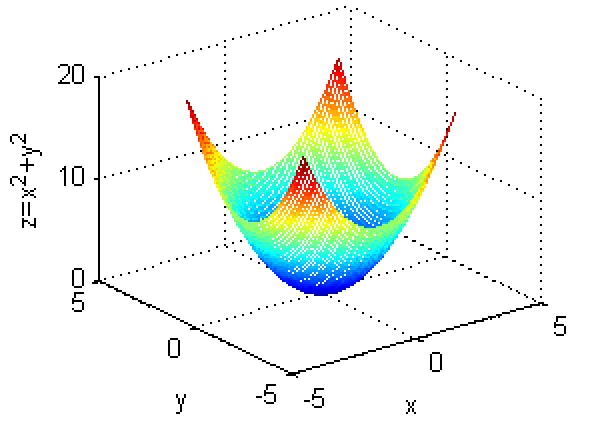
3D Graphics

We end this module on graphics with a sneak peek into 3D plots. The new functions here are `meshgrid` and `mesh`. In the example below we see that `meshgrid`produces `x`and `y`vectors suitable for 3D plotting and that `mesh(x,y,z)` plots `z` as a function of both `x` and `y`.

Example:
Example: Creating our first 3D plot.

```
[x,y] = meshgrid(-3:.1:3);
z = x.^2+y.^2;
mesh(x,y,z);
xlabel('x');
ylabel('y');
zlabel('z=x^2+y^2');
```

This code gives us the following 3D plot.


20
10
0
z=x^2+y^2
5
0
-5
y
-5
0
5
x

A Very Brief Introduction to Programming with M-Files

You can use m-file scripts to automate computations. Almost anything typed at the command line can also be included in a m-file script. Lines in a m-file script are interpreted sequentially and the instructions are executed in turn. M-file scripts allow you to implement complex computations that cannot be readily achieved using commands at the command line. You can also create computational capabilities for other people to use.

There are some differences between MATLAB, MathScript, and Octave script files; these differences are typically not that significant. M-file scripts are text files and can be edited by any text editor. The script file must have an extension of ".m" and be in a directory that MATLAB knows about. M-file names should begin with a letter and only contain letters and numbers. Any other characters (space, dash, star, slash, etc.) will be interpreted as operations on variables and will cause errors. Also, M-file names should not be the same as variables in the workspace to differentiate between file names and variables.

Note: Both MATLAB and LABVIEW MathScript have built-in editors with features that make editing m-file scripts easier. In both envrionments, the editor is integrated with a debugger which makes finding and correcting errors in your scripts easier. More detailed information about using the MATLAB editor be found at Mathworks Matlab Tutorial-Creating Scripts with MATLAB Editor/Debugger. More detailed information about using the LABVIEW MathScript editor be found at National Instruments LabVIEW MathScript Tutorial-Inside LabVIEW MathScript Tutorial..

M-file scripts interact with the current executing environment. Variables set before the script is executed can affect what happens in the script. Variables set in the script remain after the script has finished execution.

Comments document your code and help other users (and yourself several months down the road) understand how you have implemented your program. Comments begin with the character `%`; any text in a line after the `%` is ignored by the script interpreter.

To correctly execute a script, the script file environment must know the directory in which the script resides.

Note: To instruct the MATLAB environment where to search for the m-file script, you can set the current working directory or set the search path. More detailed information can be found at Mathworks Matlab Tutorial-Working with Files, Directories and Paths.

Note: To set the current working directory for LABVIEW MATHSCRIPT, use the menu **File>MathScript Preferences** in the MathScript interactive window. More detailed information can be found at National Instrument's LabVIEW MathScript Preferences Dialog Box.

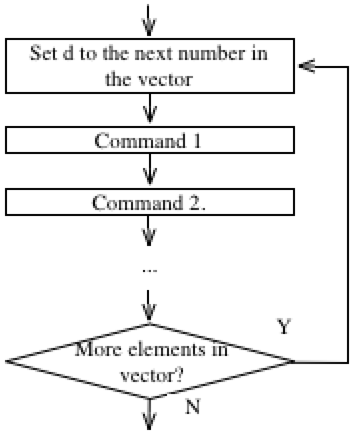
Programming with M-files: For Loops

The For Loop

The **for loop** is one way to repeat a series of computations using different values. The for loop has the following syntax: `for d = array % Command 1 % Command 2 % and so on end` In the for loop, `array` can be any vector or array of values. The for loop works like this: `d` is set to the first value in `array`, and the sequence of commands (`Command 1`, `Command 2`, `and so on`) in the body of the for loop is executed with this value of `d`. Then `d` is set to the second value in `array`, and the sequence of commands in the body of the for loop is executed with this value of `d`. This process continues through all of the values in `array`. So a for loop that performs computations for values of `d` from 1.0 to 2.0 is: `for d = 1.0:0.05:2.0 % Command 1 % Command 2 % and so on end` (Recall that `1.0:0.05:2.0` creates a vector of values from 1.0 to 2.0.)

Note that in all of the examples in this module, the commands inside the for loop are indented relative to the `for` and `end` statements. This is not required, but is common practice and makes the code much more readable.

The flow of control through a for loop is represented by the flow chart in [link]. This flow chart graphically shows how the sequence of commands in the for loop is executed once for each value. The flow of control through the for loop is also represented by the pseudo code in [link]; note that the pseudo code looks very similar to the actual m-file code.



A flow chart containing a for loop.

```
for each element of the vector
    Do Command 1
    Do Command 2
    and so on
```

Pseudo code for a for loop.

A useful type of for loop is one that steps a counter variable from 1 to some upper value: `for j = 1:10 % Commands end` For example, this type of loop can be used to compute a sequence of values that are stored in the elements of a vector. An example of this type of loop is `% Store the results of this loop computation in the vector v for j = 1:10 % Commands % More Commands to compute a complicated result v(j) = result; end`

Using a for loop to access and manipulate elements of a vector (as in this example) may be the most natural approach, particularly when one has previous experience with other programming languages such as C or Java. However, many problems can be solved without for loops by using the built-in vector capabilities. Using these capabilities almost always improves computational speed and reduces the size of the program. Some would also claim that it is more elegant.

For loops can also contain other for loops. For example, the following code performs the commands for each combination of `d` and `c`: `for d=1:0.05:2 for c=5:0.1:6 % Commands end end`

Programming with M-files: For-Loop Drill Exercises

Some For Loop Exercises

Exercise:

Problem:
Loop Indices

How many times will this program print "Hello World"? `for a=0:50 disp('Hello World') end`

Solution:

The code `0:50` creates a vector of integers starting at 0 and going to 50; this vector has 51 elements. "Hello World" will be printed once for each element in the vector (51 times).

Exercise:

Problem:
Loop Indices II

How many times will this program print "Guten Tag Welt"? `for a=-1:-1:-50 disp('Guten Tag Welt') end`

Solution:

The code `-1:-1:-50` creates a vector of integers starting at -1 and going backward to -50; this vector has 50 elements. "Guten Tag Welt" will be printed once for each element in the vector (50 times).

Exercise:

Problem:
Loop Indices III

How many times will this program print "Bonjour Monde"? `for a=-1:1:-50 disp('Bonjour Monde') end`

Solution:

The code `-1:1:-50` creates an empty vector with no elements. "Bonjour Monde" would be printed once for each element in the vector, but since the vector is empty, it is never printed.

Exercise:

Problem:
Nested Loops

How many times will this program print "Hola Mundo"? `for a=10:10:50 for b=0:0.1:1 disp('Hola Mundo') end end`

Solution:

The outer loop (the loop with `a`) will be executed five times. Each time the outer loop is executed, the inner loop (the loop with `b`) will be executed eleven times, since `0:0.1:1` creates a vector with 11 elements. "Hola Mundo" will be printed 55 times.

Exercise:

Problem:
A tricky loop

What sequence of numbers will the following for loop print? `n = 10; for j = 1:n n = n-1; j end` Explain why this code does what it does.

Solution:

In the first line, the value of `n` is set to 10. The code `1:n` creates a vector of integers from 1 to 10. Each iteration through the loop sets `j` to the next element of this vector, so `j` will be sent to each value 1 through 10 in succession, and this sequence of values will be printed. Note that each time through the loop, the value of `n` is decreased by 1; the final value of `n` will be 0. Even though the value of `n` is changed in the loop, the number of iterations through the loop is not affected, because the vector of integers is computed once before the loop is executed and does not depend on subsequent values of `n`.

Exercise:

Problem:
Nested Loops II

What value will the following program print? `count = 0; for d = 1:7 for h = 1:24 for m = 1:60 for s = 1:60 count = count + 1; end end end end count` What is a simpler way to achieve the same results?

Solution:

The `d` loop will be executed seven times. In each iteration of the `d` loop, the `h` loop will be executed 24 times. In each iteration of the `h` loop, the `m` loop will be executed 60 times. In each iteration of the `m` loop, the `s` loop will be executed 60 times. So the variable `count` will be incremented $7 \times 24 \times 60 \times 60 = 604800$ times.

A simpler way to achieve the same results is the command `7*24*60*60`

Programming With M-Files: For-Loop Exercises

Exercise:

Problem:

Frequency is a defining characteristic of many physical phenomena including sound and light. For sound, frequency is perceived as the pitch of the sound. For light, frequency is perceived as color.

The equation of a cosine wave with frequency f cycles/second is

Equation:

$$y = \cos(2\pi f t)$$

Create an m-file script to plot the cosine waveform with frequency $f = 2$ cycles/s for values of t between 0 and 4.

Exercise:

Problem:

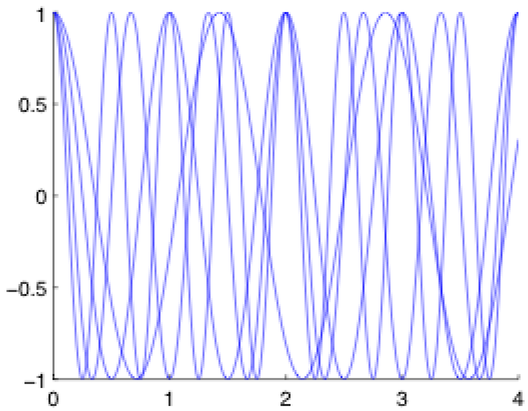
Suppose that we wish to plot (on the same graph) the cosine waveform in [link] for the following frequencies: 0.7, 1, 1.5, and 2. Modify your solution to [link] to use a for-loop to create this plot.

Solution:

The following for-loop is designed to solve this problem:

```
t=0:.01:4; hold on for f=[0.7 1 1.5 2]
y=cos(2*pi*f*t); plot(t,y); end
```

When this code is run, it plots all of the cosine waveforms using the same line style and color, as shown in [link]. The next solution shows one rather complicated way to change the line style and color.



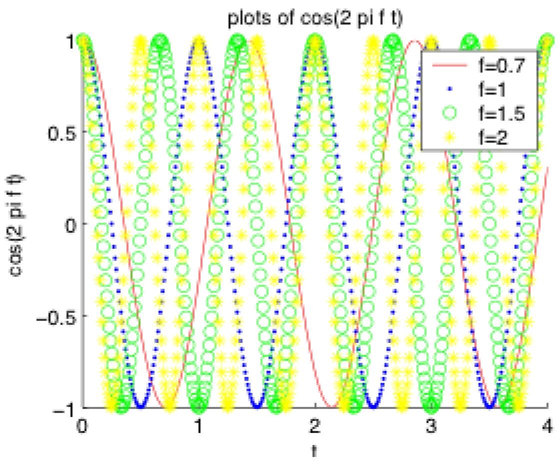
Plot of cosines at different frequencies.

Solution:

The following code changes the line style of each of the cosine plots.

```
fs = ['r-';'b.';'go';'y*']; %Create an array of line style strings x=1; %Initialize the counter variable x t=0:.01:4; hold on for f=[0.7 1 1.5 2] y=cos(2*pi*f*t); plot(t,y,fs(x,1:end)); %Plot t vs y with the line style string indexed by x x=x+1; %Increment x by one end xlabel('t'); ylabel('cos(2 pi f t)') title('plots of cos(t)') legend('f=0.7','f=1','f=1.5','f=2')
```

This code produces the plot in [link]. Note that this plot follows appropriate engineering graphics conventions-axes are labeled, there is a title, and there is a legend to identify each plot.



Plot of cosines at different frequencies.

Exercise:

Problem:

Suppose that you are building a mobile robot, and are designing the size of the wheels on the robot to achieve a given travel speed. Denote the radius of the wheel (in inches) as r, and the rotations per second of the wheel as w. The robot speed s (in inches/s) is related to r and w by the equation

Equation:

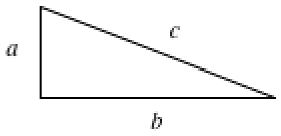
$$s = 2\pi rw$$

On one graph, create plots of the relationship between s and w for values of r of 0.5in, 0.7in, 1.6in, 3.2in, and 4.0in.

Exercise:

Problem:

Multiple Hypotenuses



Sides of a right triangle.

Consider the right triangle shown in [link]. Suppose you wish to find the length of the hypotenuse c of this triangle for several combinations of side lengths a and b; the specific combinations of a and b are given in [link]. Write an m-file to do this.

a	b
1	1
1	2
2	3
4	1
2	2

Side Lengths

Solution:

This solution was created by Heidi Zipperian: `a=[1 1 2 4 2] b=[1 2 3 1 2] for j=1:5 c=sqrt(a(j)^2+b(j)^2) end`
A solution that does not use a for loop was also created by Heidi: `a=[1 1 2 4 2] b=[1 2 3 1 2] c=sqrt(a.^2+b.^2)`

Programming with M-files: A Modeling Example Using For Loops

A Modeling Problem: Counting Ping Pong Balls

Suppose you have a cylinder of height h with base diameter b (perhaps an empty pretzel jar), and you wish to know how many ping-pong balls of diameter d have been placed inside the cylinder. How could you determine this?

> **Note:**This problem, along with the strategy for computing the lower bound on the number of ping-pong balls, is adapted from (Starfield 1994).

A lower bound for this problem is found as follows:

- N_L -Lower bound on the number of balls that fit into the cylinder.
- V_{cyl} -The volume of the cylinder.
  Equation:

$$V_{\text{cyl}} = h\pi\left(\frac{b}{2}\right)^2$$

- V_{cube} -The volume of a cube that encloses a single ball.
  Equation:

$$V_{\text{cube}} = d^3$$

The lower bound is found by dividing the volume of the cylinder by the volume of the cube enclosing a single ball.
Equation:

$$N_L = \frac{V_{\text{cyl}}}{V_{\text{cube}}}$$

Exercise:

Problem:
The interactive approach

You are given the following values:

- $d = 1.54 \text{ in}$
- $b = 8 \text{ in}$
- $h = 14 \text{ in}$

Type commands at the command line prompt to compute N_L .

Solution:

The following shows the commands typed at the >> prompt and the output produced:

```
>> d = 1.54 d = 1.5400 >> b = 8 b = 8 >> h = 14 h = 14 >> vcyl = h*pi*(b/2)^2 vcyl = 703.7168 >> vcube = d^3 vcube = 3.6523 >> nl = vcyl/vcube nl = 192.6796
```

Exercise:

Problem:
Using an M-File

Create an m-file to solve [link].

Solution:

We created the following file named `PingPong.m`:

```
% PingPong.m - computes a lower bound on the number of % ping pong balls that fit into a cylinder % Note that most lines end with a ";", so they don't print % intermediate results d = 1.54; h = 14; b = 8; vcyl = h*pi*(b/2)^2; vcube = d^3; nl = vcyl/vcube
```

When run from the command line, this program produces the following output:

```
>> PingPong nl = 192.6796
```

To complicate your problem, suppose that you have not been given values for d, b, and h. Instead you are required to estimate the number of ping pong balls for many different possible combinations of these variables (perhaps 50 or more combinations). How can you automate this computation?

One way to automate the computation of N_L for many different combinations of parameter values is to use a for loop. (Read Programming with M-Files: For Loops if you are not familiar with the use of for loops.) The following problems ask you to develop several different ways that for loops can be used to automate these computations.

Exercise:

Problem:
Use a for loop

Add a for loop to your m-file from [link] to compute N_L for $b = 8$ in, $h = 14$ in, and values of d ranging from 1.0 in to 2.0 in.

Solution:

This solution is by BrieAnne Davis.

```
for d=1.0:.05:2.0 b=8;
h=14; vcyl=h*pi*(b/2)^2 vcube=d^3 nl=vcyl/vcube
end
```

Exercise:

Problem:
Can you still use a for loop?

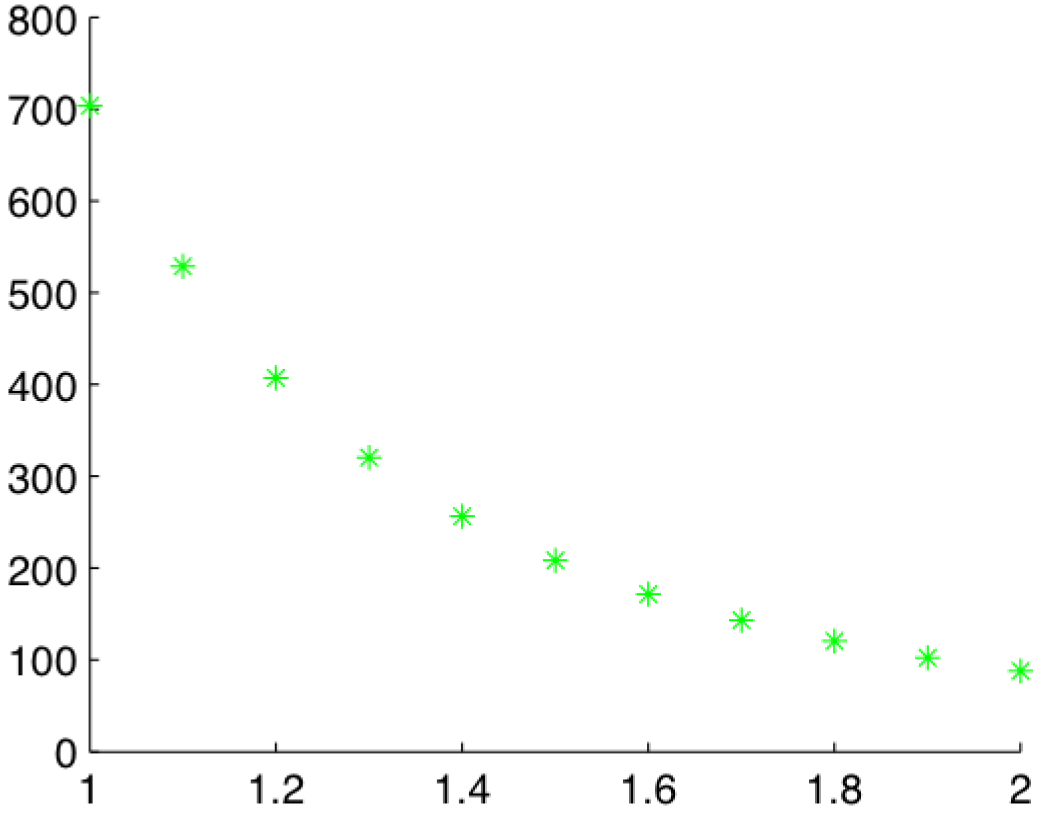
Modify your m-file from [link] to **plot** N_L as a function of d for $b = 8$ in and $h = 14$ in.

Solution:

This solution is by Wade Stevens. Note that it uses the command `hold on` to plot each point individually in the for loop.

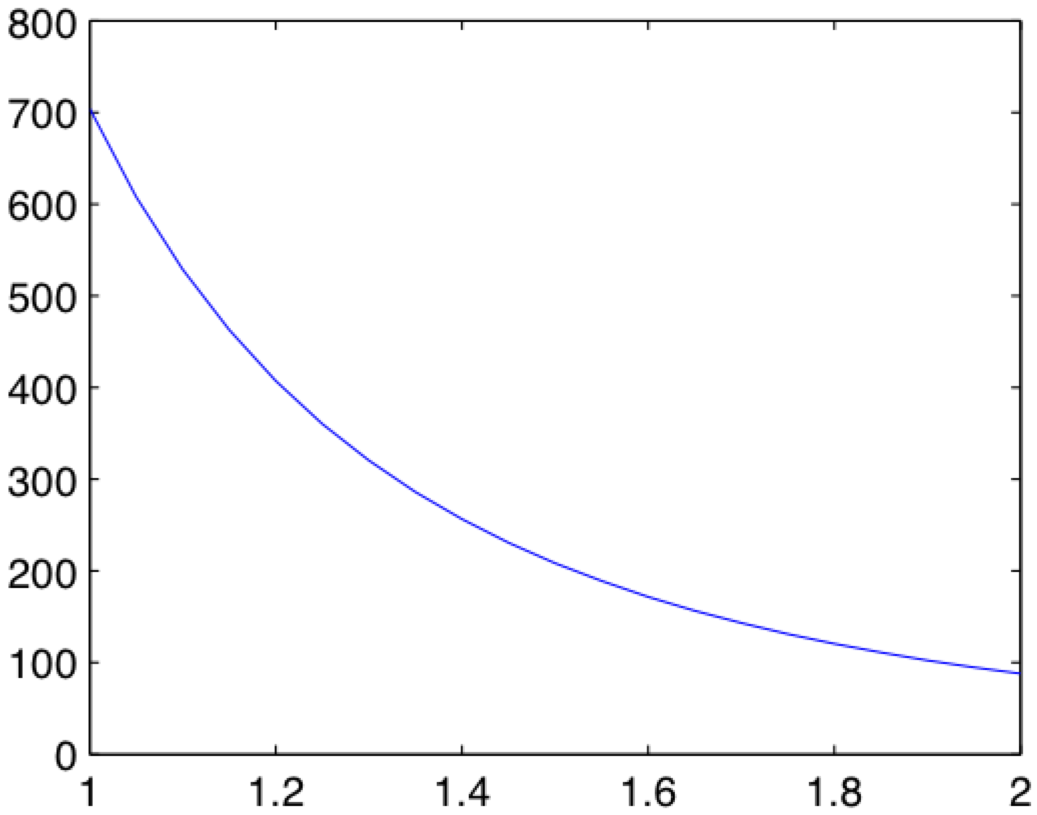
```
clear all
hold on for d=1.0:0.1:2.0; b=8; h=14; C=h*pi*
(b/2)^2; %volume of cylinder c=d^3; %volume of
```

`cube N=C/c; %Lower bound floor(N) plot (d,N,'g*') end` This solution creates the plot in [link].



Plot of N_L as a function of d; each point plotted individually.

This different solution is by Christopher Embrey. It uses the index variable `j` to step through the `dv` array to compute elements of the `nlv` array; the complete `nlv` array is computed, and then plotted outside the for loop. `clear dv=1.0:.05:2.0; [junk,dvsize] = size(dv) for j=1:dvsize d=dv(j) b=8; %in h=14; %in vcyl=h*pi*(b/2)^2; vcube=d^3; nl=vcyl/vcube; nlv(j)=nl; end plot (dv,nlv)` This solution creates the plot in [link].



Plot of N_L as a function of d; points plotted as vectors.

And finally, this solution by Travis Venson uses vector computations to perform the computation without a for loop.

```
%creates a vector
for diameter dv=1:.02:2; b=5.5; h=12; %computes
volume of cylinder vcyl=h*pi*(b/2)^2; %computes
volume of cube vcube=dv.^3; %computes lower
bound lowerboundv=vcyl./vcube; %plots results
plot(dv,lowerboundv)
```

Exercise:

Problem:
More loops?

Modify your m-file from [link] to compute N_L for $d = 1.54$ in and various values of b and h.

Solution:

This solution is by AJ Smith. The height, `h`, ranges from 12 to 15 and the base, `b`, ranges from 8 to 12.

```
for h=12:15; %ranges of height for b=8:12; %ranges of the base d=1.54; %diameter of ping pong ball. Vcyl=h*pi*(b/2)^2; %Volume of cylinder Vcube=d^3; %volume of a cube that encloses a single ball Nl=Vcyl/Vcube %lower bound on the number of balls that fit in the cylinder end end
```

Programming with M-Files: A Rocket Trajectory Analysis Using For Loops

Note: This example requires an understanding of the relationships between position, velocity, and acceleration of an object moving in a straight line. The Wikipedia article *Motion Graphs and Derivatives* has a clear explanation of these relationships, as well as a discussion of average and instantaneous velocity and acceleration and the role derivatives play in these relationships. Also, in this example, we will approximate derivatives with forward, backward, and central differences; *Lecture 4.1* by Dr. Dmitry Pelinovsky at McMaster University contains useful information about this approximation. We will also approximate integrals using the trapezoidal rule; The Wikipedia article *Trapezium rule* has an explanation of the trapezoidal rule.

Trajectory Analysis of an Experimental Homebuilt Rocket

On his web page *Richard Nakka's Experimental Rocketry Web Site: Launch Report - Frostfire Two Rocket*, Richard Nakka provides a very detailed narrative of the test firing of his Frostfire Two homebuilt rocket and subsequent data analysis. (His site provides many detailed accounts of tests of rockets and rocket motors. Some rocket launches were not as successful as the Frostfire Two launch; his site provides very interesting post-flight analysis of all launches.)

Computation of Velocity and Acceleration from Altitude Data

In this section, we will use m-files to analyze the altitude data extracted from the plot "Altitude and Acceleration Data from R-DAS" on Richard Nakk's web page. This data is in the file Altitude.txt. We will use this data to estimate velocity and acceleration of the Frostfire Two rocket during its flight.

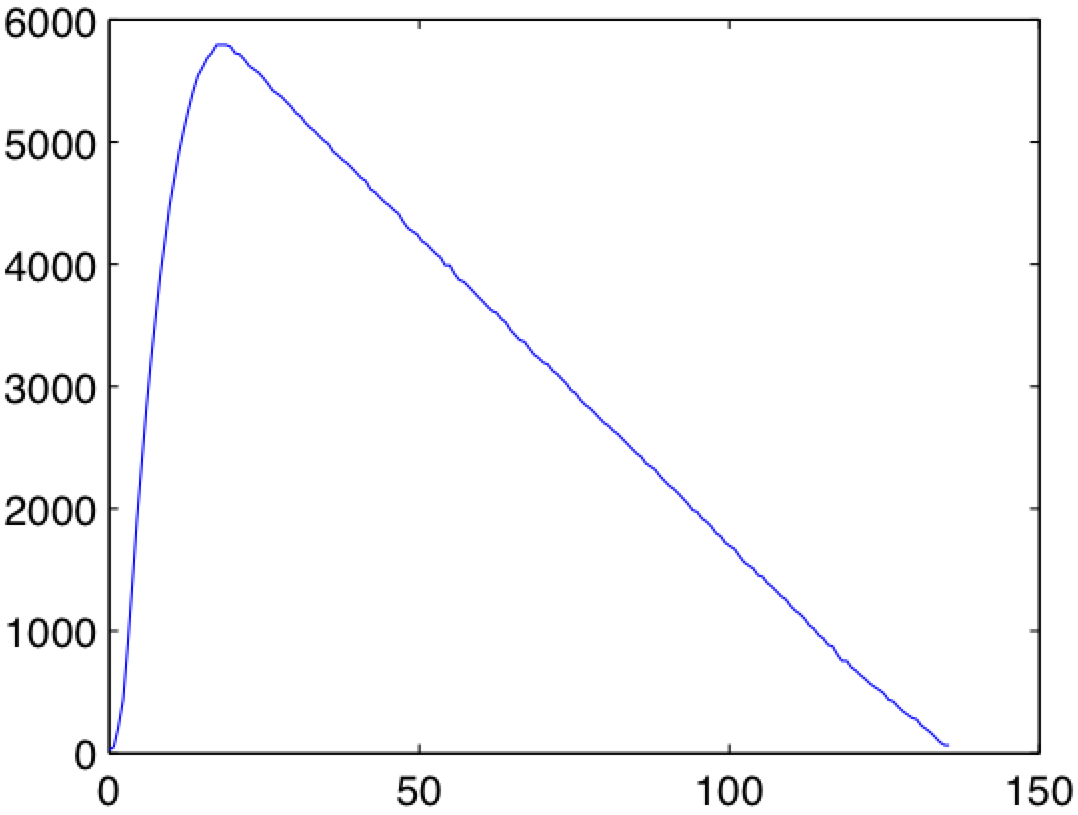
Exercise:

Problem:

Get the data

Download the altitude data set in the file Altitude.txt onto your computer (right click on this link). The file is formatted as two columns: the first column is time in seconds, and the second column is altitude in feet. Load the data and plot the altitude as a function of time.

The following sequence of commands will load the data, create a vector `t` of time values, create a vector `s` of altitude values, and plot the altitude as a function of time. `load Altitude.txt -ascii t = Altitude(:,1); s = Altitude(:,2); plot(t,s)` The plot should be similar to that in [link].



Plot of altitude versus time.

Exercise:

Problem:
Forward Differences

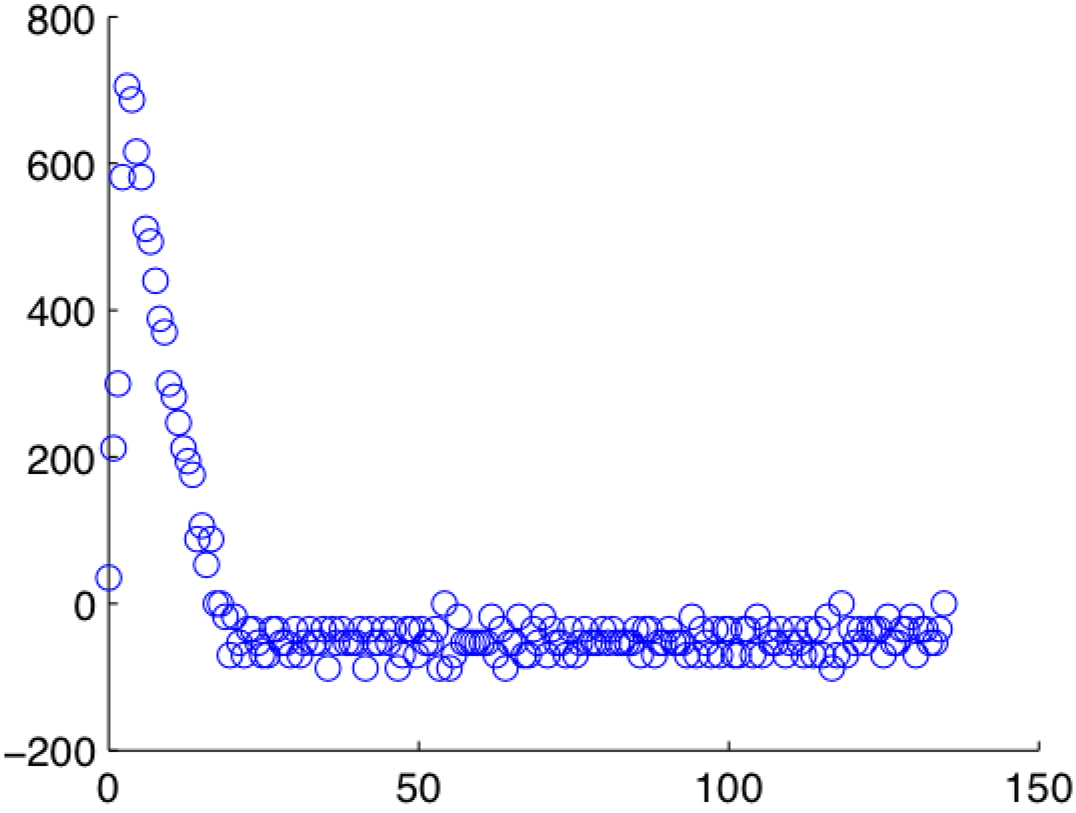
Write a script that uses a for loop to compute velocity and acceleration from the altitude data using forward differences. Your script should also plot the computed velocity and acceleration as function of time.

Solution:

This solution is by Scott Jenne; it computes and plots the velocity:

```
load Altitude.txt -ascii t=Altitude(:,1);
s=Altitude(:,2); for n=1:180; v=((s(n+1))-
s(n))/((t(n+1))-t(n)) hold on plot(t(n),v,'o')
end
```

The plot produced by this code is shown in [link].



Plot of velocity computed with the forward difference method versus time.

Exercise:

Problem:
Backward Differences

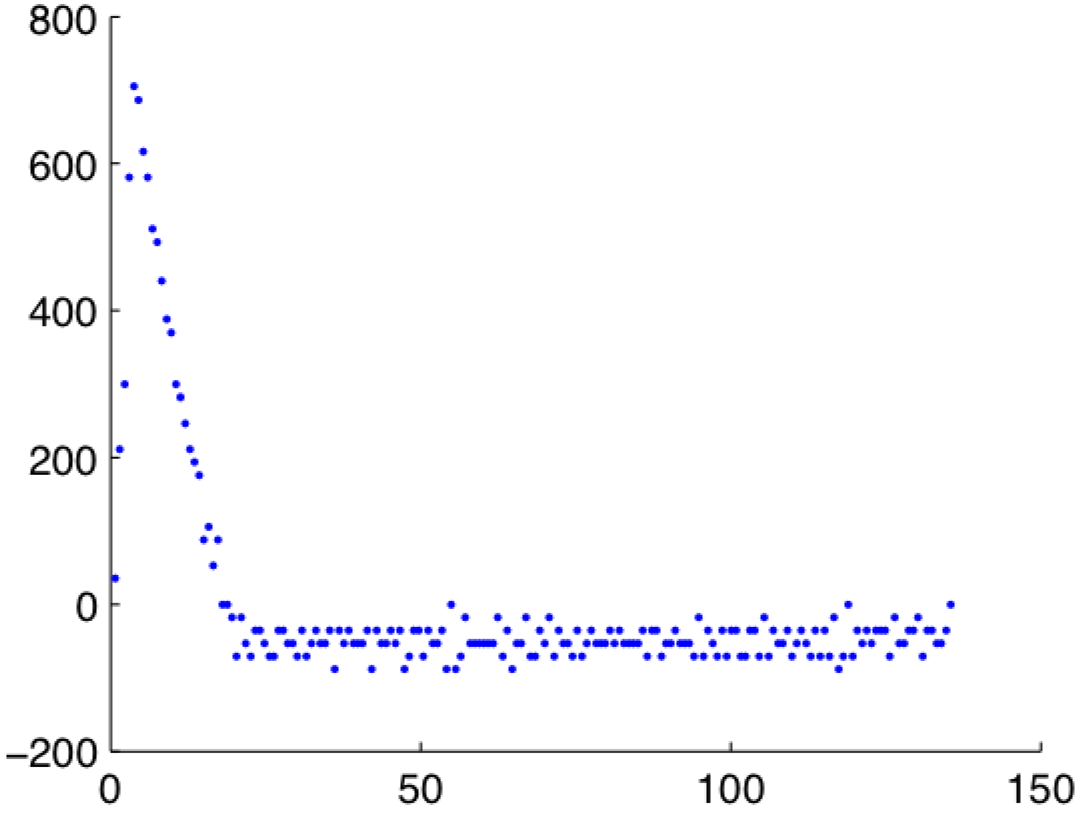
Modify your script from [link] to compute velocity and acceleration using backward differences. Remember to save your modified script with a different name than your script from [link].

Solution:

This solution by Bryson Hinton:

```
load altitude.txt -ascii
t=altitude(:,1); s=altitude(:,2); hold on for
x=2:181 v(x)=(s(x)-s(x-1))/(t(x)-t(x-1));
plot(t(x),v(x),'b.') end
```

The plot produced by this code is shown in [link].



Plot of velocity computed with the backward difference method versus time.

Exercise:

Problem:
Central Differences

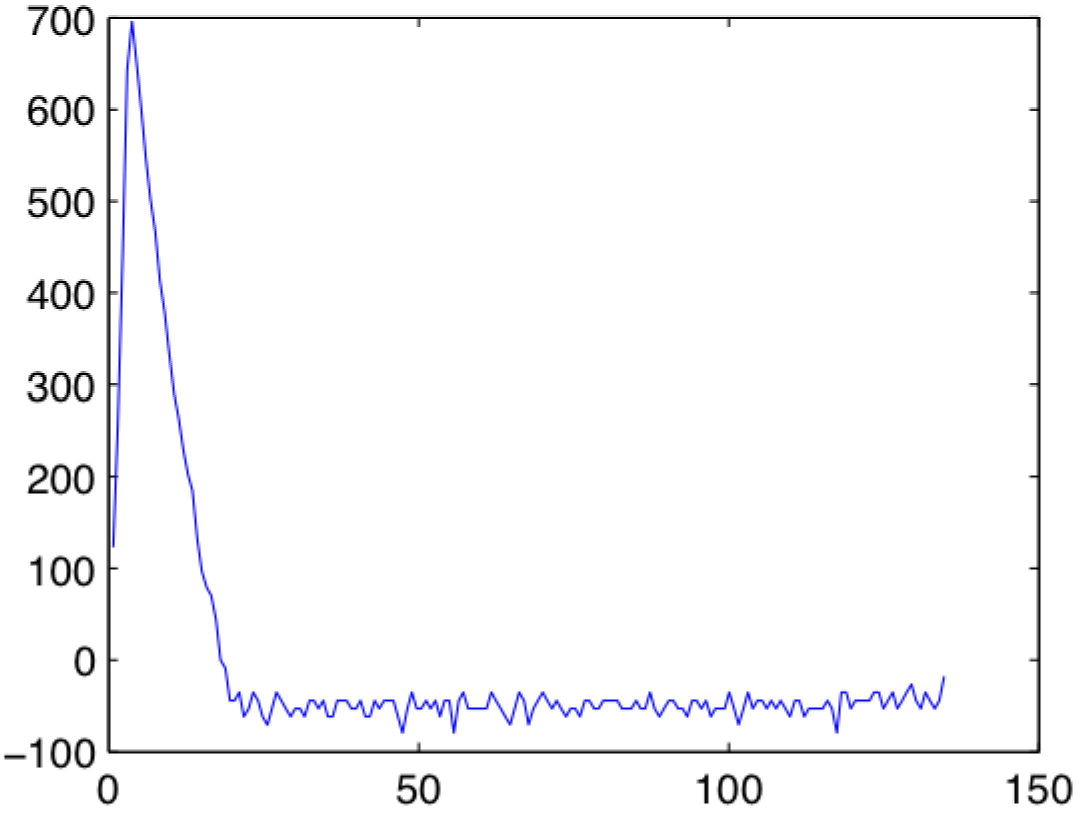
Modify your script from [link] to compute velocity and acceleration using central differences. Remember to save your modified script with a different name than your script from [link] and [link].

Solution:

This code computes the velocity using the central difference formula.

```
load Altitude.txt -ascii t=Altitude(:,1);
s=Altitude(:,2); for n=2:180 v(n-1)=(s(n+1)-
s(n-1))/(t(n+1)-t(n-1)); end plot(t(2:180),v)
```

The plot produced by this code is shown in [link].



Plot of velocity computed with the central difference method versus time.

Compare the velocity and acceleration values computed by the different approximations. What can you say about their accuracy?

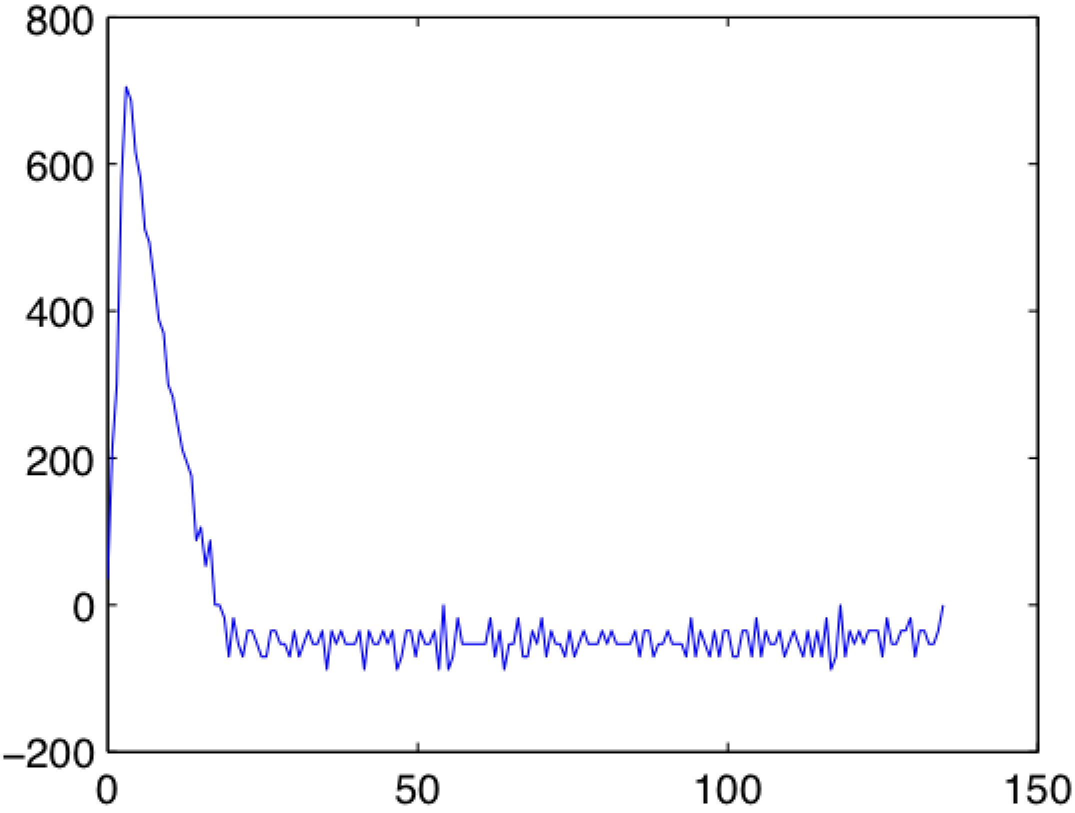
Exercise:

Problem:
Can it be done without loops?

Modify your script from [link] to compute velocity and acceleration without using a for loop.

Solution:

This code uses the `diff` function to compute the difference between adjacent elements of `t` and `s`, and the `./` function to divide each element of the altitude differences with the corresponding element of the time differences: `load Altitude.txt -ascii t=Altitude(:,1); s=Altitude(:,2); v=diff(s)./diff(t); plot(t(1:180),v)` The plot produced by this code is shown in [link].

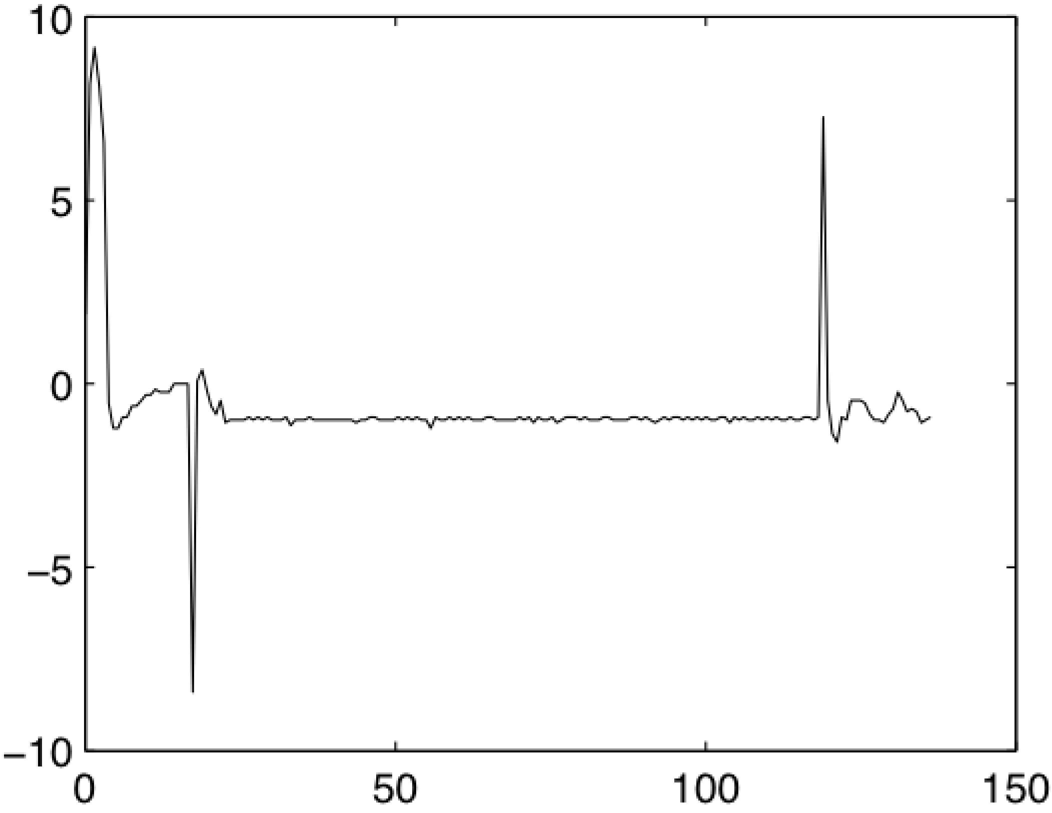


Plot of velocity computed with the forward difference method versus time. The values in this plot are the same as in [link].

Computation of Velocity and Altitude from Acceleration Data

In this section, we will use m-files to analyze the acceleration data extracted from the plot "Altitude and Acceleration Data from R-DAS" on Richard Nakk's web page. Download the acceleration data set in the file

Acceleration.txt onto your computer (right click on this link). The first column is time in seconds, and the second column is acceleration in g's. The following commands load the data and plot the acceleration as a function of time. `load Acceleration.txt -ascii t = Acceleration(:,1); a = Acceleration(:,2); plot(t,a)` The plot should be similar to that in [link].



Plot of acceleration versus time.

Exercise:

Problem:
Trapezoidal Rule

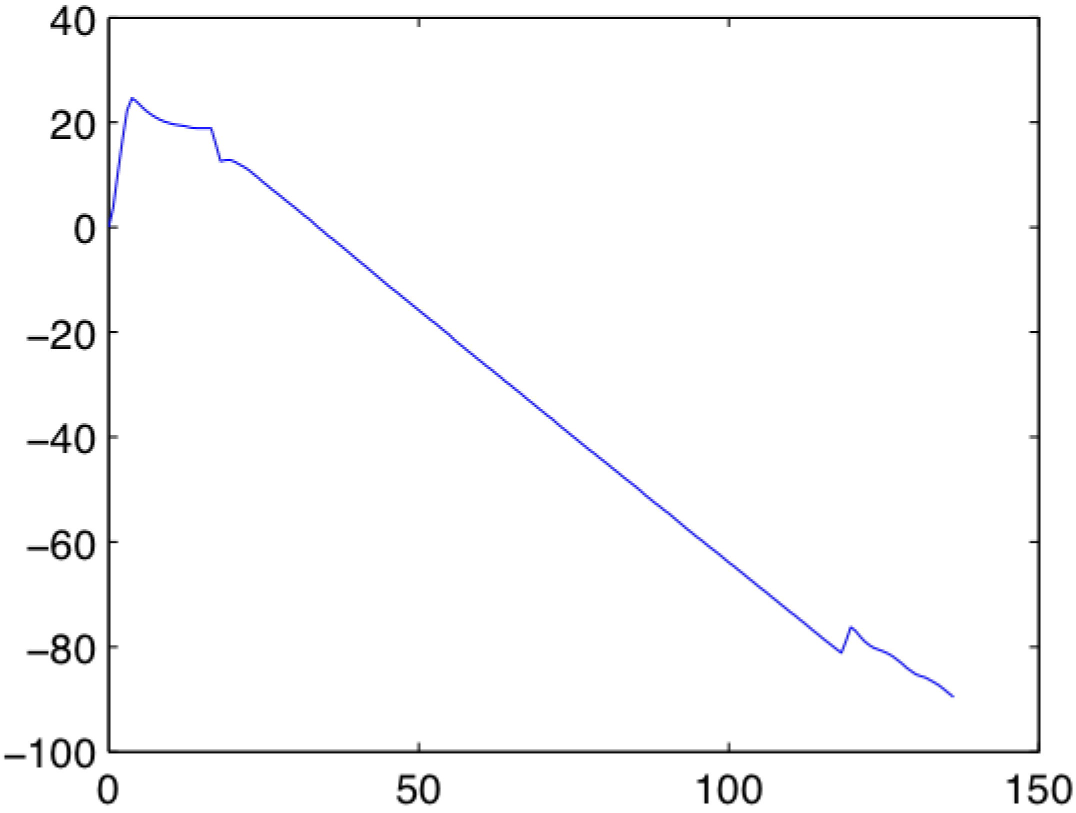
Write a script that uses a for loop to compute velocity and altitude from the acceleration data using the trapezoidal rule. Your script should also plot the computed velocity and altitude as function of time.

Solution:

This solution is by Jonathan Selby:

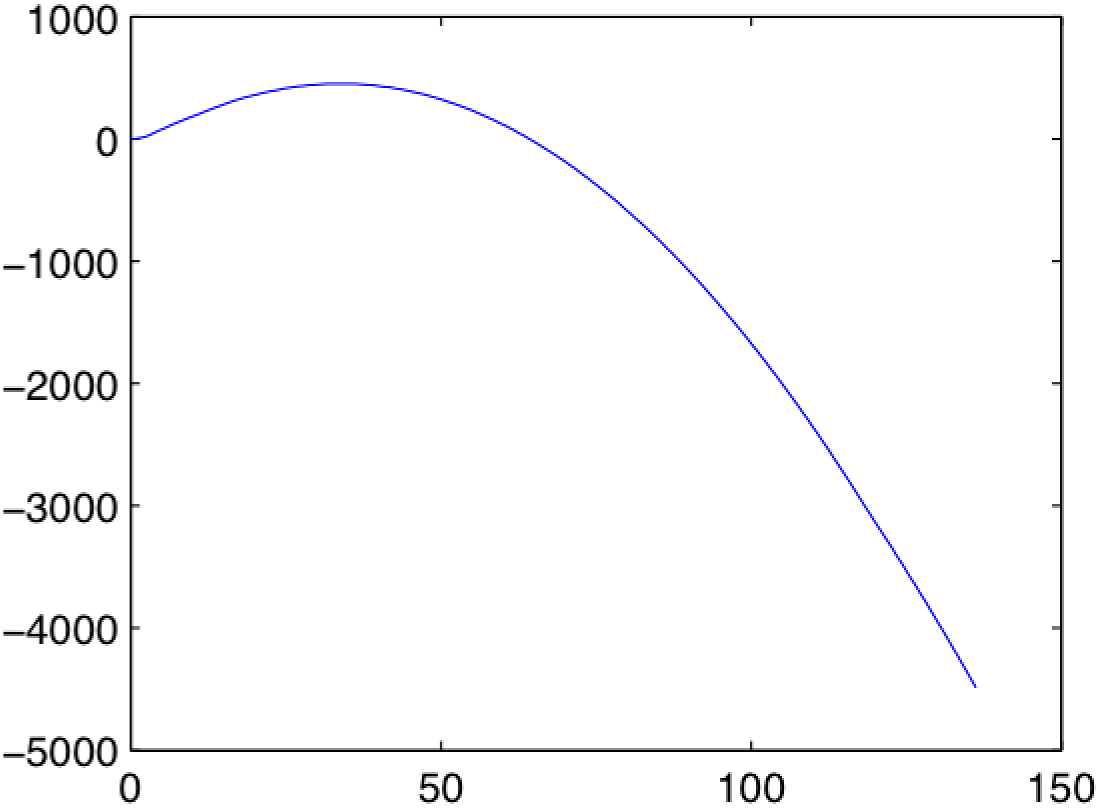
```
load Acceleration.txt -ascii t=Acceleration (:,1); a=Acceleration (:,2); v(1)=0; for n=1:181 v(n+1)=(t(n+1)-t(n))*(a(n+1)+a(n))/2+v(n); end plot(t,v)
```

This code creates the plot in [link].



Plot of velocity versus time. The velocity is computed by numerically integrating the measured acceleration.

This code can be easily extended to also compute altitude while it is computing velocity: `load Acceleration.txt -ascii t=Acceleration (:,1); a=Acceleration (:,2); v(1)=0; % Initial velocity s(1)=0; % Initial altitude for n=1:181 v(n+1)=(t(n+1)-t(n))*(a(n+1)+a(n))/2+v(n); s(n+1)=(t(n+1)-t(n))*(v(n+1)+v(n))/2+s(n); end plot(t,s)` This code creates the plot in [link].



Plot of altitude versus time.

Exercise:

Problem:
Can it be done without loops?

Modify your script from [link] to compute velocity and altitude without using a for loop.

Solution:

This solution by Nicholas Gruman uses the `cumtrapz` function to compute velocity with the trapezoidal rule: `load Acceleration.txt -ascii t=Acceleration(:,1); A=Acceleration(:,2); v=cumtrapz(t,A);` Altitude could also be computed by adding the following line to the end of the previous code: `s=cumtrapz(t,v);`

Programming with M-files: Analyzing Railgun Data Using For Loops

Note: This example requires an understanding of the relationships between acceleration and velocity of an object moving in a straight line. A clear discussion of this relationship can be found in Acceleration; the Wikipedia article *Motion Graphs and Derivatives* also has an explanation of this relationship, as well as a discussion of average and instantaneous velocity and acceleration and the role derivatives play. Also, in this example, we will compute approximate integrals using the trapezoidal rule; The Wikipedia article *Trapezium rule* has an explanation of the trapezoidal rule.

Velocity Analysis of an Experimental Rail Gun

A railgun is a device that uses electrical energy to accelerate a projectile; information about railguns can be found at the Wikipedia article *Railgun*. The paper Effect of Railgun Electrodynamics on Projectile Launch Dynamics shows the current profile of a railgun launch. The acelleration a of the projectile (in units of $\frac{m}{s^2}$) is a function of the current c through the projectile (in units of kAmp). This function is given by the equation

Equation:

$$a = 0.0036c^2 \operatorname{sgn}(c)$$

where $\operatorname{sgn}(c)$ is 1 if $c > 0$ and -1 if $c < 0$.

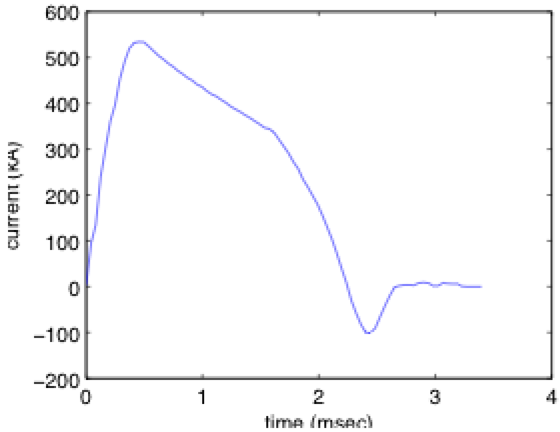
Exercise:

Problem:

Get the data

Download the current data set in the file Current.txt onto your computer (right click on this link). The file is formatted as two columns: the first column is time in mili-seconds, and the second column is current in kA.

The following sequence of commands will load the data, create a vector `t` of time values, create a vector `c` of current values, and plot the current as a function of time. `load Current.txt -ascii t = Current(:,1); c = Current(:,2); plot(t,c) xlabel('time (msec)') ylabel('current (kA)')` The plot should be similar to that in [link].



Plot of railgun current versus time.

Exercise:

Problem:

Compute the projectile velocity as a function of time. Note that velocity is the integral of acceleration.

Programming with M-Files: If Statements

The If Statement

The **if statement** is one way to make the sequence of computations executed by in an m-file script depend on variable values. The if statement has several different forms. The simplest form is `if expression % Commands to execute if expression is true end` where `expression` is a logical expression that is either true or false. (Information about logical expressions is available in Programming with M-Files: Logical Expressions.) For example, the following if statement will print "v is negative" if the variable `v` is in fact negative: `if v < 0 disp('v is negative') end`

A more complicated form of the if statement is `if expression % Commands to execute if expression is true else % Commands to execute if expression is false end` For example, the following if statement will print "v is negative" if the variable `v` is negative and "v is not negative" if `v` is not negative: `if v < 0 disp('v is negative') else disp('v is not negative') end`

The most general form of the if statement is `if expression1 % Commands to execute if expression1 is true elseif expression2 % Commands to execute if expression2 is true elseif expression3 % Commands to execute if expression3 is true ... else % Commands to execute if all expressions are false end` The following if statement is an example of this most general statement: `if v < 0 disp('v is negative') elseif v > 0 disp('v is positive') else disp('v is zero') end`

Note that in all of the examples in this module, the commands inside the if statement are indented relative to the `if`, `else`, `elseif`, and `end` statements. This is not required, but is common practice and makes the code much more readable.

Programming with M-Files: Logical Expressions

Logical Expressions

Logical expressions are used in if statements, switch case statements, and while loops to change the sequence of execution of commands in response to variable values. A logical expression is one that evaluates to either true or false. For example, `v > 0` is a logical expression that will be true if the variable `v` is greater than zero and false otherwise.

Note: In m-file scripts, logical values (true and false) are actually represented by numerical values. The numerical value of zero represents false, and any nonzero numerical value represents true.

Logical expression are typically formed using the following relational operators:

Symbol	**Relation**
`<`	Less than
`<=`	Less than or equal to
`>`	Greater than
`>=`	Greater than or equal to
`==`	Equal to

Symbol	Relation
`~=`	Not equal to

Relational Operators

Note: `==` is not the same as `=`; they are treated very differently in m-file scripting environments. `==` compares two values, while `=` assigns a value to a variable.

Complex logical expressions can be created by combining simpler logical expressions using the following logical operators:

Symbol	Relation
`~`	Not
`&&`	And
`||`	Or

Logical Operators

Programming with M-Files: If-Statement Drill Exercises

Some If Statement Exercises

Exercise:

Problem:

What will the following code print? `a = 10; if a ~= 0 disp('a is not equal to 0') end`

Solution:

' a is not equal to 0'

Exercise:

Problem:

What will the following code print? `a = 10; if a > 0 disp('a is positive') else disp('a is not positive') end`

Solution:

' a is positive'

Exercise:

Problem:

What will the following code print? `a = 5; b = 3; c = 2; if a < b*c disp('Hello world') else disp('Goodbye world') end`

Solution:

`b*c` gives a value of 6, and 5 < 6, so this code will print 'Hello world'.

Exercise:

Problem:

Suppose the code in [link] is modified by adding parentheses around `a > 0`. What will it print? `a = 10; if (a > 0) disp('a is positive') else disp('a is not positive') end`

Solution:

The parentheses around the relational expression `a > 0` will not change its validity, so this code will print 'a is positive'.

Exercise:

Problem:

Suppose the code in [link] is modified by adding the parentheses shown below. What will it print? `a = 5; b = 3; c = 2; if (a < b)*c disp('Hello world') else disp('Goodbye world') end`

Solution:

The parentheses in this expression change its meaning completely. First, `a < b` is evaluated, and since it is false for the given values of `a` and `b`, it evaluates to zero. The zero is than multiplied by `c`, giving a value of zero which is interpreted as false. So this code prints 'Goodbye world'.

Exercise:

Problem:

What will the following code print? `p1 = 3.14; p2 = 3.14159; if p1 == p2 disp('p1 and p2 are equal') else disp('p1 and p2 are not equal') end`

Solution:

' p1 and p2 are not equal'

Exercise:

Problem:

What will the following code print? `a = 5; b = 10; if a = b disp('a and b are equal') else disp('a and b are not equal') end`

Solution:

This code will generate an error message, since `a = b` assigns the value of `b` to `a`. To check if `a` and `b` are equal, use `a == b`.

Exercise:

Problem:

For what values of the variable `a` will the following MATLAB code print 'Hello world'? `if ~ a == 0 disp('Hello world') else disp('Goodbye world') end`

Solution:

Any value that is not zero.

Exercise:

Problem:

For what values of the variable `a` will the following code print 'Hello world'? `if a >= 0 && a < 7 disp('Hello world') else disp('Goodbye world') end`

Solution:

Any value greater than or equal to 0 and less than 7.

Exercise:

Problem:

For what values of the variable `a` will the following code print 'Hello world'? `if a < 3 || a > 10 disp('Hello world') else disp('Goodbye world') end`

Solution:

Any value less than 3 or greater than 10.

Exercise:

Problem:

For what values of the variable `a` will the following code print 'Hello world'? `if a < 7 || a >= 3 disp('Hello world') else disp('Goodbye world') end`

Solution:

Every value of `a` will print 'Hello world'.

Exercise:

Problem:

Write an `if` statement that will print `'a is very close to zero'` if the value of the variable `a` is between -0.01 and 0.01.

Solution:

`if a >= -0.01 && a <= 0.01 disp('a is very close to zero') end`

Programming with M-Files: An Engineering Cost Analysis Example Using If Statements

An Engineering Cost Analysis Example

Suppose you are a design engineer for a company that manufactures consumer electronic devices and you are estimating the cost of producing a new product. The product has four components that are purchased from electronic parts suppliers and assembled in your factory. You have received cost information from your suppliers for each of the parts; as is typical in the electronics industry, the cost of a part depends on the number of parts you order from the supplier.

Your assembly cost for each unit include the cost of labor and your assembly plant. You have estimated that these costs are `C0`=$45.00/unit.

The cost of each part depends on the number of parts purchased; we will use the variable `n` to represent the number of parts, and the variables `CA`, `CB`, `CC`, and `CD` to represent the unit cost of each type of part. These cost are given in the following tables.

`n`	`CA`
1-4	$16.00
5-24	$14.00
25-99	$12.70
100 or more	$11.00

Unit cost of Part A

`n`	`CB`
1-9	$24.64
10-49	$24.32
50-99	$24.07
100 or more	$23.33

Unit cost of Part B

`n`	`CC`
1-24	$17.98
25-49	$16.78
50 or more	$15.78

Unit cost of Part C

`n`	`CD`
1-9	$12.50

`n`	`CD`
10-99	$10.42
100 or more	$9.62

Unit cost of Part D

The unit cost is `Cunit = C0 + CA + CB + CC + CD`. To find the unit cost to build one unit, we look in the above tables with a value of `n`=1; the unit cost is "$45.00+$16.00+$24.64+$17.98+$12.50 = $116.12" To find the unit cost to build 20 units, we look in the above tables with a value of `n`=20 and get "$45.00+$14.00+$24.32+$17.98+$10.42 = $109.72" As expected, the unit cost for 20 units is lower than the unit cost for one unit.

Exercise:

Problem:

Create an if statement that will assign the proper cost to the variable `CA` based on the value of the variable `n`.

Solution:

```
if n >= 1 && n <= 4 CA = 16.00; elseif n >= 5
&& n <= 24 CA = 14.00; elseif n >= 25 && n <=
99 CA = 12.70; else CA = 11.00; end
```

Exercise:

Problem:

Create a script that will compute the total unit cost `Cunit` for a given value of the variable `n`.

Solution:

This code by BrieAnne Davis: `if n>=1 && n<=4; %if n=1 to 4, CA is $16.00 CA=16.00; elseif n>=5 &&`

```
n<=24; %if n=5 to 24, CA is $14.00 CA=14.00;
elseif n>=25 && n<=99; %if n=25 to 99, CA is
$12.70 CA=12.70; elseif n>=100; %if n=100 or
more, CA is $11.00 CA=11.00; end %this ends the
if statement for CA if n>=1 && n<=9; %if n=1 to
9, CB is $24.64 CB=24.64; elseif n>=10 &&
n<=49; %if n=10 to 49, CB is $24.32 CB=24.32;
elseif n>=50 && n<=99; %if n=50 to 99, CB is
$24.07 CB=24.07; elseif n>=100; %if n=100 or
more, CB is $23.33 CB=23.33; end %this ends the
if statement for CB if n>=1 && n<=24; %if n=1
to 24, CC is $17.98 CC=17.98; elseif n>=25 &&
n<=49; %if n=25 to 49, CC is $16.78 CC=16.78;
elseif n>=50; %if n=50 or more, CC is $15.78
CC=15.78; end %this ends the if statement for
CC if n>=1 && n<=9; %if n=1 to 9, CD is $12.50
CD=12.50; elseif n>=10 && n<=99; %if n=10 to
99, CD is $10.42 CD=10.42; elseif n>=100; %if
n=100 or more, CD is $9.62 CD=9.62; end %this
ends the if statement CO=45.00; Cunit=CO + CA +
CB + CC + CD;
```

Exercise:

Problem:

Create a m-file script that will compute and plot the total unit cost as a function of `n` for values of `n` from 1 to 150.

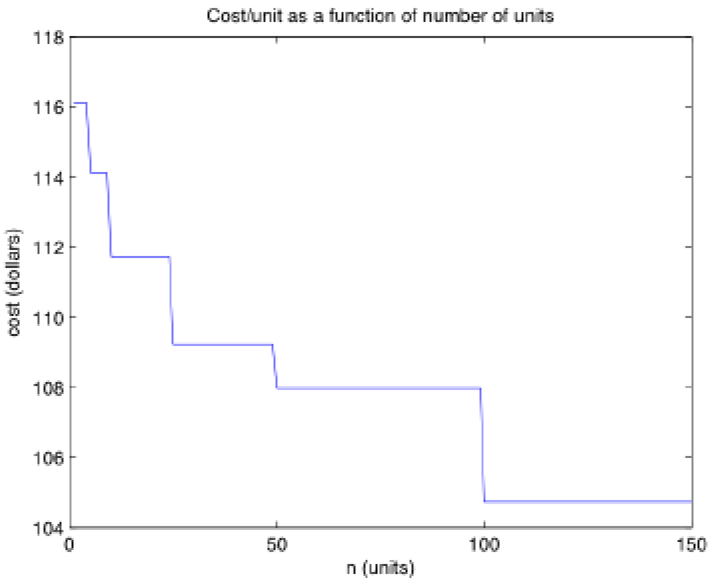
Solution:

This code was originally written by Bryson Hinton and then modified:

```
cunit = zeros(1,150); c0 = 45; for n=1:150
%compute price for part A if n >= 1 && n <= 4
ca=16; elseif n >= 5 && n <= 24 ca=14; elseif n
>= 25 && n <= 99 ca=12.7; else ca=11; end
%compute price for part B if n >= 1 && n <= 9
```

```
cb=24.64; elseif n >= 10 && n <= 49 cb=24.32;
elseif n >= 50 && n <= 99 cb=24.07; else
cb=23.33; end %compute price for part C if n >=
1 && n <= 24 cc=17.98; elseif n >= 25 && n <=
49 cc=16.78; else cc=15.78; end %compute price
for part D if n >= 1 && n <= 9 cd=12.50; elseif
n >= 10 && n <= 99 cd=10.42; else cd=9.62; end
%sum cost for all parts cunit(n)=
c0+ca+cb+cc+cd; end % Plot cost as a function
of n plot(1:150,cunit); xlabel('n (units)');
ylabel('cost (dollars)'); title('Cost/unit as a
function of number of units');
```

This code produces the plot in [link].



Cost as a function of number of units produced.

Exercise:

Problem:

Suppose that you decide to fire your workers, close down your plant, and have the assembly done offshore; in this arrangement, `C0 = Cx + Cs`, where `Cx` is the cost of offshore assembly and `Cs` is the cost of shipping assembled units from the assembler to your warehouse. After some investigation, you find an offshore assembler that gives you the following assembly costs as a function of the number of units to assemble:

`n`	`Cx`
1-29	$40.00
30-59	$30.00
60 or more	$22.00

Unit cost of Assembly

You also find a shipping company that will ship the units from the assembler to your warehouse and whose freight charges are the following function of the number of units shipped :

`n`	`Cs`
1-9	$20.00

`n`	`Cs`
10-24	$18.00
25-74	$16.00
75 or more	$15.00

Unit cost of Shipping

Update the m-file script in [link] to account for the changes in cost due to offshoring.

Programming with M-Files: While Loops

The While Loop

The **while loop** is similar to the for loop in that it allows the repeated execution of statements. Unlike the for loop, the number of times that the statements in the body of the loop are executed can depend on variable values that are computed in the loop. The syntax of the while loop has the following form: `while expression % Command 1 % Command 2 % More commands to execute repeatedly until expression is not true end` where `expression` is a logical expression that is either true or false. (Information about logical expressions is available in Programming with M-Files: Logical Expressions.) For example, consider the following while loop: `n = 1 while n < 3 n = n+1 end` This code creates the following output: `n = 1 n = 2 n = 3`

Note that in all of this example, the commands inside the while loop are indented relative to the `while` and `end` statements. This is not required, but is common practice and makes the code much more readable.

Programming with M-Files: While-Loop Drill Exercises

Some While Loop Exercises

Exercise:

Problem:

How many times will this loop print 'Hello World'? `n = 10; while n > 0 disp('Hello World') n = n - 1; end`

Solution:

10 times.

Exercise:

Problem:

How many times will this loop print 'Hello World'? `n = 1; while n > 0 disp('Hello World') n = n + 1; end`

Solution:

This loop will continue to print 'Hello World' until the user stops the program. You can stop a program by holding down the 'Ctrl' key and simultaneously pressing the 'c' key.

Exercise:

Problem:

What values will the following code print? `a = 1 while a < 100 a = a*2 end`

Solution:

`a = 1 a = 2 a = 4 a = 8 a = 16 a = 32 a = 64 a = 128`

Exercise:

Problem:

What values will the following code print?

```
a = 1; n = 1; while a < 100 a = a*n n = n + 1; end
```

Solution:

```
a = 1 a = 2 a = 6 a = 24 a = 120
```

Programming with M-Files: A Personal Finance Example Using While Loops

A Personal Finance Example

A student decides to finance their college education using a credit card. They charge one semester's tuition and then make the minimum monthly payment until the credit card balance is zero. How many months will it take to pay off the semester's tuition? How much will the student have spent to pay off the tuition?

We can solve this problem using an m-file script. We define the following variables:

- b_n - Balance at month n.
- p_n - Payment in month n.
- f_n - Finance charge (interest) in month n.

The **finance charge** $f_n$is the interest that is paid on the balance each month. The finance charge is computed using the monthly interest rate r:
Equation:

$$f_n = rb_n$$

Credit card interest rates are typically given as an annual percentage rate (APR). To convert the APR to a monthly interest rate, use the following formula:
Equation:

$$r = \left(1 + \frac{\text{APR}}{100}\right)^{\frac{1}{12}} - 1$$

More information on how to compute monthly rates can be found here.

Credit cards usually have a **minimum monthly payment**. The minimum monthly payment is usually a fixed percentage of the balance; the percentage is required by federal regulations to be at least 1% higher than the monthly interest rate. If this minimum payment would be below a given threshold (usually $10 to $20), the minimum payment is instead set to the threshold. For a threshold of $10, the relationship between the balance and the minimum payment can be shown in an equation as follows:

Equation:

$$p_n = \max\left((r + 0.01)b_n, 10\right)$$

To compute the balance for one month (month $n + 1$) from the balance for the previous month (month n), we compute the finance charge on the balance in the previous month and add it to the previous balance, then subtract the payment for the previous month:

Equation:

$$b_{n+1} = b_n + f_n - p_n$$

In the following exercises, we will develop the program to compute the number of months necessary to pay the debt. We will assume that the card APR is 14.9% (the average rate on a student credit card in mid February 2006) and that the initial balance charged to the card is $2203 (the in-state tuition at Arizona State University at the Polytechnic Campus for Spring 2006 semester).

Exercise:

Problem:

Write code to compute the monthly interest rate r from the APR using [link].

Solution:

```
APR = 14.9; r = (1+APR/100)^(1/12)-1;
```

Exercise:

Problem:

Write code to compute the minimum monthly payment p_n using [link].

Solution:

```
bn = 2203; pn = max((r+0.01)*bn,10);
```

Exercise:

Problem:

Write code to compute the balance at month $n+1$ in terms of the balance at month n using [link].

Solution:

```
fn = r*bn; bn = bn + fn - pn;
```

Exercise:

Problem:

Place the code developed for [link] into a while loop to determine how many months will be required to pay off the card.

Solution:

This space intentionally left blank.

Exercise:

Problem:

Modify your code from [link] to plot the monthly balance, monthly payment, and total cost-to-date for each month until the card is paid off.

Solution:

This space intentionally left blank.